# Concilier Expressivité et Efficacité en Programmation par Contraintes

Thierry Petit

# Concilier Expressivité et Efficacité en Programmation par Contraintes

## Dossier d'Habilitation à Diriger des Recherches

Mémoire scientifique

Thierry Petit

LINA-CNRS, INRIA, École des Mines de Nantes,

4 rue Alfred Kastler, BP 20722, F−44307 Nantes Cedex 3, France.

Foisie School of Business, Worcester Polytechnic Institute, USA.

Thierry.Petit@mines-nantes.fr, tpetit@WPI.edu

**Jury:**

| | |
|---|---|
| Christian Bessière, Dir. de Recherche CNRS, LIRMM, Montpellier, France | Rapporteur |
| Éric Monfroy, Professeur, Université de Nantes, France | Rapporteur |
| Claude-Guy Quimper, Professeur, Université de Laval, Québec, Canada | Rapporteur |
| Nicolas Beldiceanu, Professeur, Mines de Nantes, France | Examinateur |
| Narendra Jussien, Professeur, Dir. Télécom Lille, France | Examinateur |
| Pierre Lopez, Dir. de Recherche CNRS, LAAS, Toulouse, France | Examinateur |

4 décembre 2014

# Avant Propos

Depuis son apparition dans les années soixante-dix, la programmation par contraintes a suscité un intérêt croissant dans la communauté scientifique. Une des raisons de son succès est qu'il s'agit d'un paradigme théoriquement déclaratif, comme l'avait souligné Eugène Freuder :

*"Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming : The user states the problem, the computer solves it".*

Presque quarante ans après ses origines, la programmation par contraintes est largement employée pour résoudre des problèmes industriels, au détriment parfois de sa déclarativité. Il peut en effet s'avérer efficace, voire indispensable, de concevoir la modélisation du problème en exploitant des propriétés qui lui sont propres. Cet état de faits a peu à peu modifié les critères d'évaluation des contributions scientifiques, au risque parfois d'éliminer à tort certaines idées non conventionnelles ou trop novatrices pour pouvoir être employées à court terme dans un cadre industriel.

Le compromis entre généricité et efficacité se situe donc au coeur de la recherche en programmation par contraintes.

Plutôt que d'énumérer l'intégralité de mes travaux, j'ai sélectionné dans ce mémoire un ensemble de tentatives de réponse à cette problématique, ainsi que les perspectives qui s'en dégagent.

Le domaine d'application "fil rouge" illustrant mes contributions dans ce manuscrit sera l'ordonnancement sous contraintes.

Thierry Petit.

# Table des matières

# I Introduction

La Programmation Par Contraintes (PPC) est apparue dans les années soixante-dix. Le premier système générique, Alice, a été conçu par J.-L. Laurière en 1976 [84, 85]. L'article fondateur sur les réseaux de contraintes le plus ancien date de 1974, par U. Montanari [102]. Depuis, la PPC a été utilisée dans de nombreux contextes académiques et industriels, parmi lesquels l'optimisation combinatoire, l'ordonnancement, la configuration, les problèmes de simulation, la planification, la bio-informatique et la finance.

En PPC, un problème est modélisé par des variables et des contraintes. Je me suis essentiellement intéressé aux problèmes discrets, où chaque variable est munie d'un domaine fini représentant les valeurs que peuvent prendre la variable.

La PPC vise à résoudre des problèmes NP-difficiles, décrits dans un langage formel, dont les solutions sont obtenues à l'aide d'un ensemble d'algorithmes de résolution. Ces algorithmes ont la capacité de collaborer de façon générique et rigoureusement reproductible. Une contrainte exprime une relation ou propriété sur un sous ensemble de variables, c'est-à-dire, intuitivement, un sous-problème. Sachant qu'un tel sous-problème peut être commun à des applications qui n'ont rien à voir les unes avec les autres, la PPC offre par nature des outils de résolution génériques.

La PPC est :

- *efficace*, car elle est basée sur des techniques de résolution variées issues de la recherche opérationnelle et de l'intelligence artificielle,
- *modulaire*, car elle n'impose pas de limite théorique à faire communiquer des contraintes, algorithmes et techniques hétérogènes.

Le deuxième point est particulièrement important. Il a permis à la communauté de construire au fil des années des logiciels génériques puissants, intégrant des algorithmes issus d'autres disciplines de Recherche Opérationnelle et d'Intelligence Artificielle, qu'il est possible d'enrichir avec des techniques dédiées à un problème particulier.

Une des difficultés dans la conception de ces systèmes consiste à placer correctement le curseur entre, d'un côté, leur pouvoir d'expression, leur robustesse, leur simplicité (simplicité d'utilisation et simplicité du code indispensable à sa maintenance sur le long terme) et d'un autre côté l'efficacité du processus de résolution. Cette difficulté dépasse largement le cadre de l'implémentation. Elle soulève de nombreuses questions théoriques.

Plutôt que de passer en revue l'intégralité de mes travaux, j'ai sélectionné dans ce mémoire un ensemble de contributions que j'ai proposées pour répondre à cette problématique. Je présenterai pour chacune d'entre elles le positionnement dans l'état de l'art du domaine, les limites, ainsi que les perspectives qui s'en dégagent.

# II Définitions et notations

*Ce chapitre présente succinctement les notions basiques nécessaires à la lecture de ce manuscrit. Il fait l'hypothèse que le lecteur possède des connaissances minimales en mathématiques, en algorithmique et en théorie des graphes.*

## 1 Réseaux de contraintes

Un *réseau de contraintes* est défini par un ensemble de *variables* $X$, un ensemble de *domaines* et un ensemble de *contraintes*.

Ce manuscrit s'intéresse aux problèmes discrets, où chaque variable $x \in X$ est munie d'un domaine $D(x)$ de valeur entières. Selon la modélisation choisie pour un problème, le domaine d'une variable peut être représenté soit explicitement par l'ensemble des valeurs que l'on peut affecter à la variable, soit uniquement par les bornes de cet ensemble, c'est-à-dire la plus petite et la plus grande valeur de $D(x)$, respectivement notées $\underline{x}$ et $\overline{x}$.

Une contrainte $C$ porte sur un sous-ensemble de variables $Y = var(C)$ et définit une relation $rel(C)$, c'est à dire un sous-ensemble du produit cartésien des domaines des variables $var(C)$ qui spécifie les combinaisons de valeurs autorisées pour ces variables. Une telle relation peut être représentée explicitement, par la liste des combinaisons autorisées ou interdites, ou bien implicitement. On dit alors que la contrainte est définie *en intension*. La propriété sous-jacente à une définition implicite est la *sémantique* de la contrainte. Le nom de la contrainte, l'ensemble des variables $Y = var(C)$ ainsi que d'éventuels arguments permettant de définir la sémantique de la contrainte constituent sa *signature*. Nous utiliserons parfois la notation $C(Y)$. Si une affectation de valeurs est une solution de la contrainte alors cette affectation *satisfait* la contrainte. Dans le cas contraire elle viole la contrainte.

Un *réseau de contraintes* est un ensemble de contraintes.

La *solution* d'un problème de satisfaction de contraintes (CSP) est une affectation de valeurs à l'ensemble des variables du réseau, telle que chaque valeur appartient au domaine initial de sa variable − l'affectation est valide − et qui satisfait la conjonction des contraintes. Cette définition peut s'étendre au problèmes d'optimisation en définissant une ou plusieurs variables dont la valeur correspond à celle d'un critère à optimiser. Dans ce cas la qualité des solutions est mesurée par la valeur de chaque critère, ainsi que, dans de nombreux cas, par la satisfaction de contraintes dites annexes, portant sur les variables qui sont utilisées pour calculer les valeurs des critères à optimiser. Par exemple, dans un problème de minimisation des coûts de fonctionnement d'une entreprise, une contrainte annexe peut imposer une répartition équitable des coûts de fonctionnement entre les différents services.

Nous noterons $A(X)$ une affectation de valeurs à un ensemble de variables $X$. Étant donnée $x \in X$, la valeur de $x$ dans $A(X)$ sera notée $A(x)$.

## 2 Techniques de résolution

La résolution d'un problème débute par la définition d'un réseau de contraintes. Une fois la modélisation effectuée, la PPC se base sur les concepts suivants pour fournir une solution.

1. Une *méthode de parcours* de l'espace de recherche.
   Sachant que l'on vise à résoudre des problèmes difficiles, une technique d'exploration est nécessaire. Celle-ci peut être partielle, e.g., recherche à voisinage large [107], ou bien systématique. Lorsqu'elle est systématique, dans le cas de problèmes d'optimisation il est possible d'obtenir une preuve d'optimalité. Le branchement dans l'abre de recherche implicitement défini est alors dépendant d'heuristiques de choix de variables et du découpage des domaines, appelées *stratégies de recherche*.
2. Des *algorithmes de filtrage*.
   Ces algorithmes suppriment des domaines des variables des valeurs ne pouvant appartenir à une solution. Lorsque chaque algorithme de filtrage est associé à une contrainte, ce qui est le cas dans la plupart des moteurs de résolution, on emploie le terme de *propagateur*. Généralement les propagateurs sont appelés à chaque étape de l'exploration, dans un ordre défini dynamiquement par le moteur. Ces suppressions de valeurs en amont évitent de futures sélections, répétées, de ces valeurs dans la méthode de parcours. Il peut arriver que les propagateurs soient réduits à des "checkers", qui vérifient qu'un ensemble de variables dont les valeurs sont fixées satisfont ou non une contrainte. Dans ce cas il n'y a pas de suppression en amont mais une simple preuve de satisfaction.
3. Un mécanisme de *propagation*.
   Ce processus permet de communiquer les événements de suppression issus d'un algorithme de filtrage aux autres propagateurs, généralement jusqu'à l'obtention d'un point fixe.

La séparation entre modèle et résolution est essentiellement théorique. Pour obtenir un processus de résolution *efficace*, c'est-à-dire permettant d'obtenir des solutions, il est souvent nécessaire d'effectuer les choix de modélisation en fonction des algorithmes de filtrage et des stratégies de parcours.

Un propagateur peut supprimer toutes les valeurs inconsistantes avec une contrainte ou bien uniquement une partie de ces valeurs. Il est possible de caractériser le niveau de filtrage effectué au niveau de chaque contrainte et du réseau de contraintes entier. Une telle caractérisation s'appelle une *consistance*.

Nous décrivons ici les définitions utilisées dans l'état de l'art, le plus simplement possible. Notamment, les notions de contrainte et de propagateur de sont pas distinguées, c'est-à-dire, nous considérons ici exactement un propagateur associé à chaque contrainte.

Le lecteur intéressé par une description plus détaillée peut se référer au chapitre *Constraint Propagation* écrit par C. Bessière dans l'ouvrage *Handbook of Constraint Programming* [24] et à la thèse de G. Tack [145].

**Définition 1 (support, bound-support)** *Soit un ensemble de variables $X$. Étant données une variable $x$, une contrainte $C(X)$, et une valeur $v \in D(x)$.*

- *$v$ a un* support *sur $C(X)$ si et seulement si il existe une affectation valide $A(X)$ satisfaisant la contrainte $C$ telle que $A(x) = v$.*
- *$v$ a un* bound-support *sur $C(X)$ si et seulement si il existe une affectation $A(X)$ satisfaisant la contrainte $C$ telle que $A(x) = v$ et $\forall y \in X$, $y \neq x$, $A(y) \in [\underline{y}, \overline{y}]$.*

**Définition 2 (GAC)** *Une contrainte $C(X)$ est "Generalized Arc-Consistent" (*GAC*) si et seulement si $\forall x \in X$, $\forall v \in D(x)$, $v$ a un support sur $C(X)$. Un réseau de contraintes est GAC si toutes les contraintes qui le composent sont GAC.*

**Définition 3 (BC)** *Une contrainte $C(X)$ est "Bound-Consistent" (*BC*) si et seulement si $\forall x \in X$, $\underline{x}$ et $\overline{x}$ ont un bound-support sur $C(X)$. Un réseau de contraintes est BC si toutes les contraintes qui le composent sont BC.*

**Définition 4 (RC)** *Une contrainte $C(X)$ est "Range-Consistent" (*RC*) si et seulement si $\forall x \in X$, $\forall v \in D(x)$, $v$ a un bound-support sur $C(X)$. Un réseau de contraintes est RC si toutes les contraintes qui le composent sont RC.*

La RC peut-être vue comme une consistance intermédiaire entre la consistance aux bornes BC et la consistance d'arc GAC. Elle est peu utilisée car pour de nombreuses contraintes le coût opérationnel pour l'établir s'avère équivalent ou proche à celui de la GAC.

En pratique, il est naturellement possible d'appliquer des niveaux de consistance différents d'une contrainte à une autre.

En outre, il existe des consistances dites *fortes*, prenant en compte plus d'une contrainte pour déterminer la viabilité d'une valeur. Nous aborderons brièvement ce dernier point dans le chapitre IV.

## 3 Ordonnancement cumulatif

Dans ce manuscrit j'utiliserai le plus souvent possible des exemples issus de la même famille d'applications : les problèmes d'ordonnancement. Ce choix de présentation a pour but de faciliter la compréhension des liens existant entre mes différentes contributions.

Les problèmes d'ordonnancement revêtent une importance majeure dans de nombreux problèmes industriels, notamment lorsque l'objectif est de réduire les coûts de fonctionnement,

de production ou de maintenance. Ce choix m'a donc semblé particulièrement pertinent dans le contexte économique actuel.

Un problème d'ordonnancement consiste à ordonner des *activités* dans le temps, également appelées *tâches* lorsqu'on ne se préoccupe pas de la quantité de ressource qu'elles requièrent ou bien qu'elles produisent. Nous nommons $\mathcal{A}$ l'ensemble de ces activités. Un des problèmes d'ordonnancement les plus couramment étudiés est le problème cumulatif.

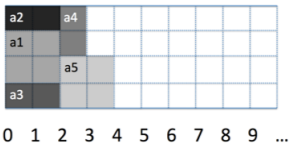


FIGURE 1 – Solution d'un CuSP minimisant le *makespan*, avec $C = 4$.

Dans un problème cumulatif (CuSP), les activités $a \in \mathcal{A}$ ne peuvent pas être interrompues et ont une durée $p_a$. Elles nécessitent pour leur exécution une quantité de ressource $h_a$. Sans perte de généralité nous pouvons considérer que $h_a$ et $p_a$ sont des entiers, car en programmation par contraintes la généralisation au cas de variables est triviale pour la plupart des algorithmes de résolution dans le cas d'un problème pur. $s_a$ et $e_a$ sont les variables de début et de fin d'une activité $a$. La ressource disponible en tout point de temps est limitée par la capacité $C$. Ce problème est NP-Difficile [105].

**Définition 5 (CuSP)** *Soit un ensemble d'activités $\mathcal{A}$ ne peuvent pas être interrompues. Une solution au problème cumulatif* CuSP *est un ordonnancement des activités qui satisfait les contraintes suivantes.*

$$\forall a \in \mathcal{A}, s_a + p_a = e_a \wedge \forall t \in \mathbb{N}, (\sum_{\substack{a \in \mathcal{A}, \\ t \in [s_a, e_a[}} h_a) \leq C$$

Le plus souvent, un problème d'optimisation est résolu. Par exemple, un critère usuel consiste à minimiser le *makepan*, c'est-à-dire, la date de fin d'activité la plus tardive.

# III Ordonnancement robuste

*Une problématique au coeur du compromis entre expressivité et efficacité est la gestion de l'incertitude dans les problèmes d'optimisation. Comment obtenir des solutions robustes à des événements perturbants ? Quelles sont les définitions de la notion de robustesse qui correspondent aux besoins des problèmes concrets ? Quelles sont les techniques de résolution les mieux adaptées ?*
*Ce chapitre présente un cadre de travail original, car il se démarque des approches probabilistes de simulation, ou encore à la construction de solutions communes à plusieurs instances d'un même problème. Il est basé sur l'idée que l'on peut intégrer la robustesse aux aléas dans la définition du problème et utiliser des outils de PPC classiques pour le résoudre. Les techniques de résolution prennent alors en compte la sémantique du problème. Nous avons étudié les problèmes d'ordonnancement cumulatifs robustes et démontré la capacité de la PPC a intégrer une notion de robustesse dans une application réelle comportant des contraintes métier non triviales.*

*Collaborations : PhD. Alban Derrien, Dr. Stéphane Zampelli.*
*Publications majeures : article accepté à CP2014* [50].

## 1 Contexte et problématique

De nombreuses applications requièrent une prise en compte de facteurs d'incertitude. Nous nous intéressons plus particulièrement aux problèmes d'ordonnancement où une garantie de robustesse aux aléas est requise *a priori*.

**Applications ciblées.** Le point de départ de cette étude a été le suivant : dans de nombreuses applications, il n'est pas envisageable de produire une nouvelle solution lorsqu'un événement perturbant se produit. Cette impossibilité est due non seulement à la difficulté de résolution des problèmes traités, mais aussi et surtout à des contraintes industrielles. Dans un tel contexte, les techniques de *réparation* ne sont pas bien adaptées.

Certains problèmes portuaires [154] présentent cette caractéristique. Les employés responsables de la planification du déchargement des containers dans les bateaux ont besoin quotidiennement d'une solution fixée, qui garantit que le déchargement se terminera avant une échéance précise, sous peine de pénalités financières importantes. D'un autre côté, l'expertise des ouvriers qui manipulent les grues est variable, ainsi que les conditions météorologiques, notamment le vent et les facilités d'accès aux cargos. Il est donc nécessaire de fournir des solutions respectant l'échéance mais qui conservent un certain degré de robustesse aux aléas. Les opérateurs sont en mesure d'évaluer les risques : les besoins de résistance à l'incertitude *doivent* être pris en compte en tant que données, pour chaque activité.

Un facteur de robustesse aux incertitudes d'exécution peut être obtenu de manière empirique, en ajoutant des activités fictives qui pourront fournir la marge de manœuvre nécessaire en cas

d'événements imprévus. Cette approche n'est pas satisfaisante : le facteur de robustesse de la solution n'est pas formellement quantifiable et il n'y a pas de garantie que certaines étapes du chargement n'en soient pas dépourvues.

Par conséquent, il existe un besoin industriel d'approches déclaratives pour traiter la robustesse qui soient spécialisées aux problèmes d'ordonnancement sous contraintes. Le paradigme proposé sera réaliste s'il satisfait deux conditions incontournables dans ce type d'applications :

1. L'approche doit être *performante* : l'ajout de robustesse ne doit pas engendrer une trop forte dégradation de l'objectif, qui dans le cas des ports est la date de fin de la dernière activité de chargement. Cette date doit se situer le plus tôt possible, avant l'échéance fixée par les opérationnels.
2. L'approche doit être *modulaire* : les problèmes ne sont pas purs, les techniques proposées doivent donc rester valides lorsqu'on impose de respecter des contraintes métier annexes au problème d'ordonnancement central.

La PPC étant par nature modulaire, il m'a semblé intéressant d'étudier en profondeur cette problématique dans le cadre du problème CuSP avec minimisation du makepsan, c'est-à-dire, la date de fin de la dernière activité. Ce problème correspond aux besoins logistiques existant dans le domaine applicatif décrit précédemment. Ce travail de recherche a été réalisé en collaboration avec Alban Derrien, doctorant sous ma direction scientifique, et Stéphane Zampelli, expert des problèmes d'optimisation dans l'industrie maritime.

**État de l'art.** En PPC, E. Hébrard et al. ont proposé une approche générique pour la robustesse dans les réseaux de contraintes, basée sur la notion de *super-solution* [61, 62]. Dans une $(a, b)$-super-solution, le changement de la valeur d'au plus $a$ variables peut être réparé en changeant les valeurs d'au plus $b$ variables. Cette approche, parce qu'elle est générique, est telle que toutes les variables ont le même statut. La sémantique du problème n'est pas prise en compte. En ordonnancement, elle a été appliquée sur des benchmarks de job-shop. La propriété intéressante de ce paradigme est sa déclarativité : le facteur de robustesse des solutions est formellement quantifié, en amont de la résolution. Nous avons poursuivi cette voie.

Une alternative plus spécifique à l'ordonnancement, un peu naïve, consiste à augmenter la durée toutes les activités. L'ajout, appelé *slack*, pourra être utilisé pour gérer à un événement inattendu. Cette méthode étant peu performante dans le cas d'une minimisation du makespan, car toutes les activités sont allongées. Elle a été améliorée en incorporant dans les algorithmes de résolution, en cours de recherche, un raisonnement sur l'incertitude [44]. Cette technique est intéressante mais ne correspond pas aux besoins que nous avons ciblés, d'une part car l'incertitude n'est pas traitée comme une donnée mais de façon probabiliste, d'autre part parce que les problèmes abordés sont des problèmes de job-shop.

D'autres approches existent en Recherche Opérationnelle pour les problèmes de job-shop et d'open-shop, ainsi que des variantes plus spécifiques à des familles d'applications [33, 153].

Enfin, concernant le problème CuSP, une approche MIP (Mixed-Integer Linear Programming) a été proposée pour traiter des problèmes robustes de RCPSP (c'est-à-dire, CuSP avec en plus des contraintes de précèdence entre activités) [93]. La formulation requiert un nombre exponentiel de variables et de contraintes, et doit donc être approximée pour obtenir des solutions.

## 2 Une nouvelle approche pour les problèmes cumulatifs robustes

Nous définissons un nouveau problème cumulatif, appelé $R_1$CuSP, tel que dans n'importe quel ensemble de fenêtres de temps disjointes un *slack* soit disponible pour une des activités dans la fenêtre. Cette définition est une spécialisation de la notion de super-solution, intégrant la sémantique du problème cumulatif.

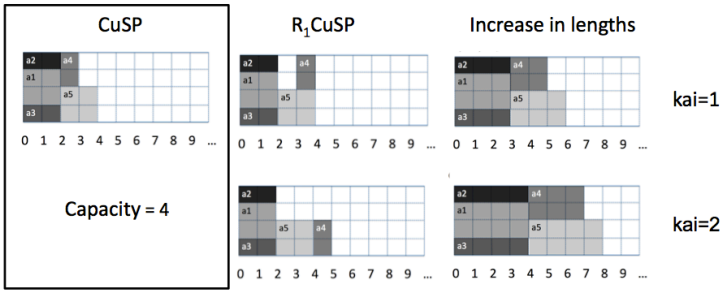


FIGURE 2 – Solutions optimales (minimisation du makespan) de problèmes cumulatifs. L'axe horizontal est le temps, l'axe vertical est la consommation en ressource. Les rectangles gris représentent des activités. À gauche, un CuSP classique, sans notion de robustesse. Au milieu, les solutions optimales du problème $R_1$CuSP, tel que toutes les valeurs dans $\mathcal{K}$ soient respectivement égales à 1 (haut) et 2 (bas). À droite, les solutions optimales dans le cas où toutes les activités sont augmentées respectivement de 1 et 2 unités de temps.

**Définition 6 ($R_1$CuSP)** *Soit $\mathcal{A}$ un ensemble d'activités et $\mathcal{K}$ un ensemble d'entiers positifs associés avec les activités, tel qu'à chaque activité $a \in \mathcal{A}$ correspond $k_a \in \mathcal{K}$. Une solution du problème $R_1$CuSP satisfait les contraintes suivantes :*

$$\forall a \in \mathcal{A}, s_a + p_a = e_a$$

$$\wedge \quad \forall t \in \mathbb{N}, \Big(\sum_{\substack{a \in \mathcal{A}, \\ t \in [s_a, e_a[}} h_a\Big) + \Big(\max_{\substack{a \in \mathcal{A}, \\ t \in [e_a, e_a + k_a[}} h_a\Big) \leq C$$

La définition 6 considère un ensemble $\mathcal{K}$ d'entiers positifs pour représenter les slacks. Il est possible, sans restriction, d'utiliser des variables pour les slacks. Les algorithmes de résolution seront les mêmes à l'exception qu'u lieu de considérer un slack entier fixé il conviendra de considérer la plus

petite valeur du domaine de la variable de slack. Avec des variables, des contraintes additionnelles peuvent être ajoutées sur les slacks, notamment lorsque leur longueur dépend du moment où est exécutée l'activité.

Nous pouvons exprimer un problème $R_1$CuSP à l'aide d'une contrainte, comme cela est fait pour les CuSP. Sa signature est la suivante : FlexC($\mathcal{A}, C, \mathcal{K}$).

**Performance.** Le problème $R_1$CuSP peut être généralisé au cas $R_r$CuSP, où dans des fenêtre de temps distinctes $r$ activités $a$ peuvent être retardées ou décalées de $k_a$ points de temps sans engendrer de modifications sur l'ordonnancement. Si $r$ est trop grand alors cela revient à augmenter la durée de toutes les activités, c'est-à-dire, à la méthode naïve.

Lorsque l'on cherche à minimiser le makespan, $R_1$CuSP est le compromis le plus performant entre le facteur de robustesse et la performance. Nous avons conjecturé et vérifié en pratique que ce compromis est pourtant tout à fait suffisant en termes de robustesse. D'une part, il est possible de retarder plusieurs activités dans des fenêtres de temps différentes. D'autre part, plus d'une activité peut être modifiée dans certaines fenêtres. C'est le cas des activités $a_1$ et $a_2$ dans la figure 2. Ces cas particuliers sont d'autant plus fréquents que le makespan est, de façon raisonnable, souvent plus élevé que dans le problème CuSP classique.

**Modularité.** Sachant que l'on représente le problème par une contrainte, notre approche est naturellement modulaire. Cependant, les contraintes additionnelles portant sur des débuts d'activité doivent elles aussi être robustes. Nous avons démontré sur un exemple pratique de chargement de bateaux qu'il est possible de considérer des problèmes avec de nombreuses contraintes métier tout en conservant un bon facteur de performance. Ces résultats sont présentés un peu plus loin.

**Algorithme de filtrage.** Nous avons adapté une technique de filtrage de type Time-Table à la contrainte FlexC, issue de l'algorithme d'A. Letort et al. pour CuSP [92, 91]. Nous avons choisi cet algorithme, appelé Sweep Dynamique, car il permet de passer à l'échelle sur de gros problèmes (plusieurs milliers d'activités). Il existe un mode glouton permettant même de traiter des problèmes allant jusqu'à un million d'activités, mais cet algorithme casse la modularité (on ne peut alors plus définir de contraintes annexes). Nous n'avons donc adapté que le propagateur.

Notre algorithme étant assez différent de l'originel et un peu plus complexe (notamment, le cas des bornes inférieures des domaines des variables de début d'activité et le cas des bornes supérieures ne sont pas symétriques), nous avons démontré expérimentalement que son passage à l'échelle reste compétitif avec l'algorithme.

Nous avons comparé l'approche à un réseau de contraintes exprimant un $R_1$CuSP à l'aide de $n$ contraintes Cumulative. Nous avons prouvé que cette décomposition effectue le même filtrage que notre algorithme.

Nous avons reproduit des expériences similaires à celles de la thèse d'A. Letort [91]. Nous avons

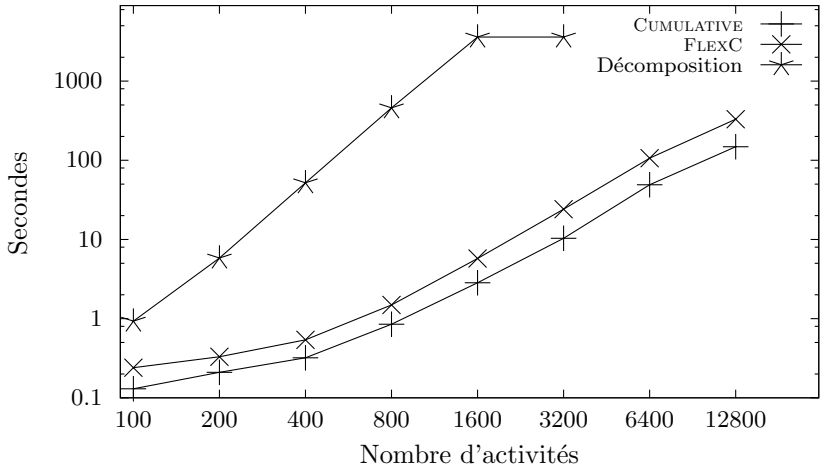


Figure 3 – Passage à l'échelle de l'algorithme de Sweep Dynamique pour FlexC.

généré des instances avec des activités telles que la durée $p_a$ varie de 5 à 10, la consommation $h_a$ de 1 à 5, et la capacité est égale à $C = 30$. Le but est de trouver une première solution. Ces problèmes ne sont pas combinatoires, mais ils mettent en défaut, dans le cas du CuSP, tous les propagateurs existants à l'exception de celui d'A. Letort dès que le nombre d'activités dépasse le millier. Les valeurs générées dans $\mathcal{K}$ ne sont pas nulles, avec une moyenne égale à 4. Des résultats équivalents sont obtenus en fixant la même valeur pour tous les $k_a$.

La Figure 3 montre que notre algorithme résout des problèmes jusqu'à 12800 activités, là où la décomposition dépasse une limite d'une heure pour des problèmes de 1600 activités, et provoque un dépassement de mémoire avec 6400 Cumulative (sur un Appel Mac, OSX 10.8.5, processeur 2.9 Ghz Intel i7 avec 8GB of RAM).

## 3 Application aux problèmes portuaires

Le problème CAP (Crane Assignment Problem) est une spécialisation du problème Berth & Crane [154] où l'on traite l'ordonnancement détaillé du déchargement des containers d'un cargo.

Un cargo est composé de zones qui correspondent à des sections transverses de stockage des containers. Chaque zone est découpée en deux parties, inférieure et supérieure. Un nombre fixé de grues est affecté au bateau, qui se déplacent sur un unique rail. Elles ne peuvent pas se croiser. Le but est de minimiser le temps nécessaire au déchargement du cargo, car le terminal paie une taxe proportionnelle au temps de stationnement du cargo. Le déchargement dépend des conditions météorologiques (vent et mer), des ouvriers pilotant le grue et de la position des containers dans le cargo. De nombreuses contraintes métier régissent les activités : précédences, temps de transition, affectation des machines, etc.

Pour éviter des pénalités financières, les clients souhaitent une planification fixe qui dans son

pire cas d'exécution, selon un certain facteur de robustesse aux aléas, offre la garantie de se terminer avant une échéance précise.

Naturellement, le problème n'est pas pur. Des contraintes spécifiques doivent être prises en compte. Nous avons démontré qu'il est possible de transformer les contraintes métier de sorte à les rendre compatibles avec la notion de robustesse définie via le problème $R_1$CuSP.

**Protocole expérimental.** Nous avons traité des problèmes à 20 zones par cargo et 4 grues. Les activités (min. deux par zones) ont des durées variant de 5 à 800 minutes, avec un facteur de robustesse (slack, $k_a$) simulé aléatoirement pour chaque activité correspondant à un pourcentage entre 0 et 25% de sa durée. Ces données sont réalistes pour des bateaux de taille moyenne.

La technique d'exploration utilisée est une recherche à voisinage large (LNS) guidée par la propagation [107], avec une remise en cause aléatoire de 30% des activités et une stratégie de choix de la variable de début ayant le plus petit domaine, affectée avec la date la plus petite d'abord. Nous avons fixé une limite de temps de cinq minutes.

Nous avons mesuré la distance entre le makespan et une borne inférieure qui ignore les contraintes métier additionnelles, dans le cas d'un CuSP classique, d'un modèle robuste utilisant FlexC et d'un modèle robuste naïf où le slack est ajouté en dur à la durée de chaque activité.

La colone $\alpha$ donne la position relative de la valeur d'objectif de l'approche robuste (FlexC) comparée au makespan du CuSP et de l'approche naïve. La colone $\gamma$ est le ratio entre le modèle naïf et celui utilisant FlexC. Une valeur $\gamma = 2$ indique que le modèle naïf double la distance à la borne inférieure théorique, par rapport au modèle utilsant FlexC.

| # | CAP | FlexC | Modèle naïf | $\alpha$ | $\gamma$ |
|---|---|---|---|---|---|
| 1 | 10.2 (0.08) | 20.7 (2.20) | 36.2 (3.28) | 0.40 | 2.47 |
| 2 | 8.6 (0) | 19.4 (2.13) | 32.5 (1.62) | 0.45 | 2.21 |
| 3 | 5.7 (0) | 19.2 (3.08) | 28.5 (0.60) | 0.59 | 1.68 |
| 4 | 9.3 (0) | 17.7 (0.47) | 30.9 (3.66) | 0.38 | 2.57 |
| 5 | 6.3 (0) | 20.2 (2.10) | 35.3 (0) | 0.47 | 2.08 |
| 6 | 6.4 (0) | 17.8 (1.37) | 30.7 (0.15) | 0.46 | 2.13 |
| 7 | 4 (0) | 15.5 (1.08) | 35.8 (2.40) | 0.36 | 2.76 |
| 8 | 1.8 (0) | 19.8 (2.35) | 27.1 (1.42) | 0.71 | 1.40 |
| 9 | 13.2 (0) | 15.5 (2.61) | 22 (0.31) | 0.26 | 3.82 |
| 10 | 7.2 (0) | 19.4 (1.63) | 31.6 (0.07) | 0.5 | 2.0 |

Table 1 – Distance en pourcentages à une borne inférieure théorique du temps de déchargement.

**Résultats.** Notre approche produit une solution robuste ajoutant 10% en moyenne à la distance à la borne théorique, par rapport à un modèle cumulatif classique sans robustesse. Ce pourcentage reste compatible avec les échéances fixées en pratique, contrairement au résultat du modèle naïf qui ajoute 20% en moyenne. La borne inférieure théorique du temps de déchargement d'un cargo totalement plein est de (20*2*800)/4=8000 minutes, soit 5 jours, 13 heures et 20 minutes. Le

modèle naïf ajoute plus d'un jour (26 heurs et 40 minutes). Le modèle utilisant FlexC ajoute 13 heures et 20 minutes, ce qui constitue un bon compromis.

## 4 Perspectives

Le traitement de l'incertitude revêt une importance qui est au coeur de la problématique du compromis nécessaire entre efficacité et expressivité. L'aspect intéressant, particulièrement en ordonnancement, est que les besoins en termes d'expressivité se déclinent en concepts qui sont difficiles à concilier : robustesse aux aléas et qualité des solutions (à la fois concernant les contraintes métier et la performance des solutions relativement aux critères d'optimisation).

Nos travaux ont montré l'intérêt de traiter un problème d'optimisation sous incertitude dans le cadre classique de la PPC, en exploitant une définition de la notion de robustesse corrélée au problème par l'intermédiaire d'une contrainte globale. Cette preuve de concept permet d'envisager d'employer le même type d'approches pour d'autres classes d'applications en optimisation sous contraintes. Notre contribution étant récente (2014), il n'existe pas d'article scientifique qui l'exploite ou l'étende.

# IV Aide à la modélisation

*L'expressivité de la PPC cache une difficulté d'utilisation qui la rend inaccessible à de nombreux utilisateurs dits "non experts". Il ne suffit pas de modéliser un problème, car il existe très souvent plusieurs modèles possibles pour un même problème. Il convient de définir le réseau de contraintes en fonction des algorithmes présents dans le moteur de résolution que l'on utilise, éventuellement d'ajouter de nouvelles contraintes, de définir des heuristiques de recherche spécialisées, etc. Une des perspectives majeures de la communauté scientifique est de rendre les outils de PPC concrètement déclaratifs. J'ai abordé cette problématique selon trois axes :*

1. Apprentissage.
   *Exploiter les propriétés d'un modèle fourni par l'utilisateur supposé ne pas être un expert en PPC, via un mécanisme d'apprentissage, peut permettre d'améliorer automatiquement ce modèle. Notre approche a été la première à prendre en compte, en PPC, non seulement la structure du réseau mais aussi les domaines des variables. Elle s'adapte à chaque instance.*
2. Techniques de reformulation et de représentation générique des contraintes.
   *L'idée est de pouvoir générer automatiquement un propagateur lorsque l'on fournit une représentation d'une contrainte selon un certain formalisme, dans notre cas les formalismes du catalogue de contraintes de Beldicanu et al.* [14]. *Nous étudions aussi la notion de reformulation dans le cadre du filtrage, en proposant une technique de ré-ordonnancement des valeurs dans les domaines permettant d'utiliser des propagateurs de BC pour réaliser un filtrage plus fort, e.g., GAC.*
3. Caractérisation formelle des propagateurs et consistances alternatives.
   *Cette contribution est basée sur l'idée qu'il existe des contraintes pour lesquelles de nombreux propagateurs ont été proposés, dont les caractéristiques ne peuvent se résumer aux simples notions de GAC, BC ou RC. D'une part la puissance du filtrage obtenu peut ne correspondre à aucune de ces trois notions. D'autre part il existe d'autres métriques intéressantes pour classifier des propagateurs. Sans une caractérisation formelle plus précise, le choix du propagateur ne peut être fait que statiquement par un utilisateur expert en PPC, ce qui constitue un frein considérable à la diffusion de la PPC.*

   *En outre, nous avons travaillé sur l'intégration des consistances plus fortes que GAC dans les moteurs de résolution événementiels. L'intérêt de cette contribution est d'ouvrir ces techniques à l'utilisation de l'ensemble des contraintes disponibles dans les moteurs, ainsi qu'aux contraintes "utilisateur".*

*Collaborations : Pr. Nicolas Beldiceanu, Dr. Hab. Chirstian Bessière, Pr. Mats Carlsson, Dr. Romuald Debruyne, Dr. Sophie Demassey, Dr. Jean-Guillaume Fages, Pr. Narendra Jussien, Dr. Guillaume Rochart, Dr. Julien Vion, Dr. Hab. Bruno Zanuttini.*
*Publications majeures : CP2005 (2), CP2006, IJCAI2007, IJCAI2009, ECAI2012, Constraints (2), RAIRO, LNAI (CSCLP2009).*

# 1 Apprentissage de Contraintes Implicites

Une des barrières s'opposant à une diffusion plus ample de la programmation par contraintes est l'expertise importante requise pour concevoir un modèle permettant une résolution efficace d'un problème. Aussi, l'intérêt de la communauté pour la reformulation automatique des modèles à contraintes a été très important ces dix dernières années [41, 127, 57, 80, 1]. Nous proposons une approche alternative [25, 26], consistant à apprendre des contraintes implicites d'après des instanciations des variables du problème, solutions et non-solutions. Ces contraintes sont alors ajoutées au modèle afin d'améliorer la résolution. Pour valider l'impact de cette approche, nous proposons un algorithme d'apprentissage de contraintes de cardinalité (GlobalCardinality [132]).

Nous montrons expérimentalement l'intérêt de cet algorithme pour améliorer la qualité des modèles à contraintes dans un contexte réel : l'amélioration automatique de modèles écrits en Choco par des étudiants de l'IUT de Montpellier.

## 1.1 Principe

Le processus de modélisation d'un problème débute en général par la conception d'un premier modèle, qui exprime le problème à l'aide de contraintes et de variables. Si le temps de résolution s'avère trop élevé pour passer à l'échelle, une des techniques employées pour améliorer le modèle est l'ajout de contraintes *implicites* [142] : ces contraintes n'apportent rien du point de vue de la modélisation, mais elles ne modifient pas l'ensemble des solutions du problème. Ce principe est illustré par la figure 4.

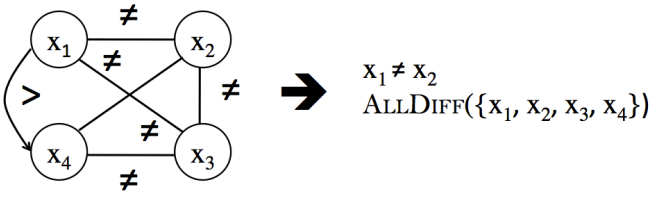


Figure 4 – Un réseau de contraintes et des contraintes implicites. La contrainte AllDiff impose que toutes les variables impliquées prennent des valeurs différentes. Elle n'est pas explicitement présente dans le réseau de contraintes initial, à gauche. L'ajouter au modèle peut améliorer le processus de résolution car elle traite globalement le problème alors que les contraintes du réseau initial sont binaires (deux variables).

Si les propagateurs associés à ces contraintes permettent un gain significatif en terme de suppressions de valeurs inconsistantes, il est important de les ajouter de façon explicite au réseau de contraintes (voir par exemple l'article de Régin présentant le détail du processus de modélisation d'un problème de Sport League Scheduling [133]).

Nous avons proposé un paradigme d'apprentissage automatique de contraintes implicites [25,

26]. L'idée initiale est la suivante : si une contrainte portant sur un sous-ensemble de variables $X$ du réseau est implicite, alors poser sa négation supprime toutes les solutions du problème. Prouver l'inconsistance d'un problème est certes généralement plus coûteux que de trouver une solution, mais dans notre contexte il est possible de faire cette preuve uniquement sur le réseau de contraintes réduit au sous-ensemble $X$. Dans le meilleur des cas, le sous-problème peut même s'avérer polynomial à résoudre, par exemple si le réseau de contraintes obtenu est Berge-acyclique [23, 70, 71].

### 1.2 Originalité de l'approche

Nous nous sommes focalisés sur des contraintes dites "à paramètres", c'est à dire telles qu'un ensemble de variables, une relation ou encore un ensemble d'entiers régissent une propriété à satisfaire sur les variables de décisions. Il existe de très nombreuses contraintes de ce type, telles que les *soft global constraints* [122] ou encore les contraintes de cardinalité (e.g., GlobalCardinality [132], NValue [106], etc.). Pour ces contraintes, nous avons extrait les propriétés théoriques favorables à un processus d'apprentissage rapide. Nous invitons le lecteur intéressé par cette caractérisation formelle à consulter notre article le plus général sur le sujet [26].

L'originalité et l'intérêt majeur de cette approche, relativement à l'ensemble des techniques existant dans l'état de l'art, est que ce fut la première à prendre en compte les domaines des variables. Les contraintes apprises sont donc spécifiques à chaque instance de problème, ce qui est plus précis qu'une approche structurelle, qui ne raisonne que sur les contraintes et les variables.

Un autre aspect original est qu'au fur et à mesure du processus d'apprentissage, la processus de preuve d'inconsistance des problèmes avec la négation d'une contrainte s'avère de plus en plus aisé, car on ajoute au modèle les contraintes implicites précédemment apprises. En d'autres termes, le processus d'apprentissage est naturellement incrémental.

### 1.3 Mise en œuvre

J'ai codé en Choco 1.2 un moteur d'apprentissage basé sur ce principe, spécialisé à la contrainte GlobalCardinality, car cette contrainte est présente de façon implicite dans un grand nombre d'instances de problèmes, et parce qu'elle possède toutes les bonnes propriétés pour obtenir un processus d'apprentissage rapide.

En collaboration avec C. Bessière et R. Coletta, nous avons confronté ce moteur au cas concret de modèles à contraintes réalisés par des utilisateurs non experts de la PPC. Dans cette expérience, nous avons repris l'implémentation d'un problème d'affectation d'options à des élèves, réalisés par un groupe d'étudiants de Montpellier. Nous n'avons réalisé aucune autre modification de leur modèle que l'ajout de contraintes GlobalCardinality apprises grâce au moteur. Les résultats ont été satisfaisants, comme l'indiquent la table 2.

| $n$ | Modèle initial | | Ajout de contraintes implicites | |
|---|---|---|---|---|
| | #nœuds | résolution | #nœuds | apprentissage + résolution |
| 10 | 110 | **0.2 s** | 12 | **0.2 s + 0.0 s** |
| 20 | 137,982 | 183.2 s | 1,998 | **10.2 s + 8.4 s** |
| 60 | — | $>$ 12 heures | $8.7 \times 10^6$ | **110.0 s + 43 minutes** |

TABLE 2 – Apprentissage de GLOBALCARDINALITY implicites en partant du code de projets d'étudiants sur un problème d'affectation d'options à des élèves. Les temps (résolution et apprentissage) sont indiqués en secondes.

### 1.4 Portée des travaux

Il s'est avéré difficile de caractériser formellement les critères de découpage du réseau en sous-problèmes permettant de générer des contraintes implicites améliorant la résolution, sans s'appuyer sur la sémantique du problème. Ce point constitue une perspective intéressante. Néanmoins, notre approche ayant été pionnière en PPC concernant le fait d'intégrer les spécificités de chaque instance au processus d'apprentissage de contraintes implicites, elle ouvre une voie nouvelle par rapport aux travaux existants.

Dans la littérature, une technique visant les mêmes objectifs, étendue à d'autres types de contraintes a été présentée à la conférence CP [86]. Récemment, Leo et al. [89] ont poursuivi le même objectif en utilisant des méthodes différentes, appelées "globalization techniques".

## 2 Représentations génériques des contraintes

Dans le cadre de l'aide à la modélisation, il peut être intéressant d'obtenir un propagateur lorsque l'on fournit une contrainte reformulée dans un certain formalisme, par exemple à partir d'automates [9] ou encore à partir d'une représentation des contraintes par des propriétés de graphe [21, 22, 10, 11]. En effet, en composant blocs génériques inhérents à chaque représentation, il peut être possible de générer un propagateur. Cette génération peut être automatisée.

### 2.1 Bornes de propriétés de graphes

N. Beldiceanu a proposé une représentation originale des contraintes, basée sur des propriétés de graphe [6].

Intuitivement, on peut présenter cette technique comme une représentation d'une contrainte par un réseau de contraintes dynamique, par opposition aux décompositions classiques. Dans cette nouvelle forme de décomposition, au début de la recherche certaines contraintes ont un statut indéterminé : il n'est pas possible de trancher sur le fait qu'elles doivent ou non être satisfaites. Leur satisfaction dépend de propriétés imposées sur le graphe de contraintes. Ces propriétés expriment la sémantique de la contrainte initiale qui est représentée. Au fur et à mesure de la résolution, les domaines sont réduits et les informations émanant des propriétés sont de plus en plus précises.

Nous avons étudié l'idée, avec N. Beldiceanu et G. Rochart [21], d'obtenir des bornes sur les valeurs des propriétés, avec l'idée de fournir des composants logiciels (checkers ou propagateurs) génériques basés sur les propriétés de graphe. En collaboration avec N. Beldiceanu, M. Carlsson et S. Demassey, nous avons étudié les propagateurs issus des propriétés les plus communes [10]. Sachant que le nombre de propriétés utiles est somme toute très restreint (une quinzaine), cette approche fournit une technique pour générer automatiquement un propagateur pour plusieurs centaines de contraintes, qu'il est possible de décrire à l'aide de cette représentation.

**Properiétés de graphe.** Une contrainte globale $C$ est représentée par un graphe orienté initial $G_i = (X_i, E_i)$ : à chaque sommet de $X_i$ correspond une variable impliquée dans $C$, et à chaque arc $e$ de $E_i$ correspond une contrainte binaire impliquant les variables extrémités de $e$. Pour générer $G_i$ à partir des paramètres de $C$, on utilise l'ensemble de générateurs d'arcs décrits dans [6].

Lorsque toutes les variables de $C$ sont fixées, on supprime de $G_i$ toutes les contraintes binaires qui ne sont pas satisfaites, ainsi que les sommets isolés, c'est-à-dire, les sommets qui ne sont pas l'extrémité d'un arc. Ce graphe final est noté $G_f$. $C$ est définie par une conjonction de propriétés de graphes qui doivent être satisfaites par $G_f$. Chaque propriété de graphe a la forme $P$ *op* $V$ ; $P$ est une caractéristique du graphe, $V$ une variable et *op* est un opérateur de comparaison $\geq, \leq, =, \neq$. Dans le catalogue de contraintes [14], les propriétés les plus communément appliquées au graphe final $G_f$ sont basées sur les caractéristiques suivantes :

- **NARC** et **NVERTEX**, qui désignent le nombre d'arcs et de sommets.
- **NCC** et **NSCC**, le nombre de composantes connexes et fortement connexes.
- **NSINK** et **NSOURCE**, le nombre de sommets n'ayant aucun successeur et aucun prédécesseur.

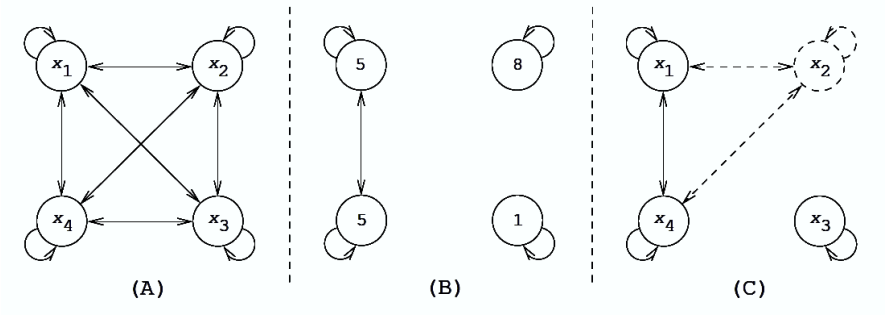


FIGURE 5 – (A) Graphe initial $G_i$ associé à la contrainte NVALUE$(N, \{x_1, x_2, x_3, x_4\})$. (B) Graphe final $G_f$ de la solution NVALUE$(3, \{5, 8, 1, 5\})$. (C) Graphe intermédiaire. Figure extraite de l'article référencé en [21].

Considérons par exemple la contrainte NVALUE($N$,$X$) [106]. Cette contrainte impose que le nombre de valeurs distinctes présentes dans l'ensemble de variables $X$ soit égal à la valeur de la variable $N$.

Les parties (A) et (B) de la figure 5 montrent respectivement le digraphe initial $G_i$ généré pour la contrainte `nvalue` avec $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ et le digraphe $G_f$ associé à la solution `nvalue`$(3, \{5, 8, 1, 5\})$. Chaque sommet de $G_i$ correspond à une variable. Tous les arcs qui correspondent à des contraintes d'égalité non satisfaites sont supprimés afin d'obtenir $G_f$ d'après $G_i$. La figure montre pour chaque sommet de $G_f$ la valeur qui a été affectée à la variable correspondante. La contrainte NVALUE est définie par la propriété de graphe $\mathbf{NSCC} = N$. NVALUE$(3, \{5, 8, 1, 5\})$ est satisfaite car $G_f$ contient trois composantes fortement connexes, qui peuvent être interprêtées comme le fait que $N$ soit égal au nombre de valeurs distinctes affectées aux variables $x_1, x_2, x_3$ et $x_4$.

**Dériver un propagateur des propriétés.** D. Hanák a réalisé une première exploitation de cette description pour définir des algorithmes de filtrage, pour une propriété de graphe particulière [60]. Dans [21, 22, 10], nous présentons une approche systématique ayant pour objectif de produire des algorithmes de filtrage pour les propriétés de graphes les plus usuelles.

Il est tout d'abord nécessaire d'introduire la notion de *graphe intermédiaire* issu du graphe initial $G_i$, et tel que les sommets et les arcs ont différents états. Le rôle de ce graphe intermédiaire est de refléter la connaissance courante que l'on a des sommets et des arcs de $G_i$ qui pourront appartenir ou ne pas appartenir au graphe final $G_f$. Cette connaissance est issue de deux sources :

- à cause du domaine courant de ses deux variables, une contrainte binaire associée à un arc de $G_i$ n'est pas satisfaite (ou doit obligatoirement être satisfaite),
- à cause d'une raison externe un arc ou un sommet de $G_i$ doit impérativement appartenir à $G_f$ (ou ne doit impérativement pas appartenir à $G_f$).

Lorsqu'une contrainte globale $C$ est posée le graphe intermédiaire correspond à $G_i$, tandis que quand toutes les variables de $C$ ont été instanciées, le graphe intermédiaire est égal à $G_f$.

Etant donnée une propriété de graphe $P$ *op* $V$ associée à une contrainte globale $C$, le graphe intermédiaire sera utilisé pour évaluer une borne inférieure $\underline{P}$ et une borne supérieure $\overline{P}$ de la caractériqtique de graphe $P$.

Ces bornes inférieures sont évaluées à l'aide d'algorithmes de résolution de problèmes classiques en théorie des graphes [58], parfois à l'aide d'un graphe transformé, obtenu à partir de $G_i$. Ces algorithmes peuvent être simples, ou bien plus complexes (par exemple la cardinalité d'un couplage maximal dans un graphe), voir NP-Difficiles (par exemple, la cardinalité d'un hitting set minimal).

À partir du calcul de ces bornes, il est possible de vérifier que la contrainte peut encore être satisfaite : si ça sémantique implique un résultat sur la propriété qui sort des bornes calculées avec

| Theorem Parts | Complexity | Graph Related Problems |
|---|---|---|
| **Theorem 1** | | |
| • Triggering | NP-hard [10] | *cardinality of a minimum hitting set* |
| • Item 1 | $O(m)$ | *iterating through the arcs* |
| • Items 2,4 | NP-hard ? | *identifying vertices that do not belong to any minimum hitting set* |
| • Item 3 | NP-hard ? | *identifying vertices that belong to every minimum hitting set* |
| **Corollary 1** | | |
| • Triggering | $O(n)$ | *iterating through the vertices* |
| • Item 1 | $O(n)$ | *iterating through the vertices* |
| **Theorem 2** | | |
| • Triggering | $O(n)$ | *iterating through the vertices* |
| • Item 1 | $O(n)$ | *iterating through the vertices* |
| **Theorem 3** | | |
| • Triggering | $O(n)$ | *iterating through the vertices* |
| • Item 1 | $O(m)$ | *iterating through the arcs* |
| • Items 2,3 | $O(m)$ | *depth first search* |
| **Theorem 4** | | |
| • Triggering | $O(m\sqrt{n})$ | *maximum cardinality matching* [12] |
| • Item 1 | $O(m)$ | *computing the cc* |
| • Items 2,5,6 | $O(m \cdot n)$ | *identifying edges that do not belong to any maximum cardinality matching* [14] |
| • Items 3,4 | $O(m)$ | *identifying vertices that are saturated in every maximum cardinality matching* [5] |

FIGURE 6 – Tableau extrait de l'article CP2006 [10] : Théorèmes de filtrage pour les propriétés de graphes du catalogue de contraintes.

les domaines courants, alors un échec est détecté. De même, si une valeur dans le domaine d'une variable aboutit à cet état, alors on peut la supprimer du domaine.

**Originalité et portée des travaux.** Cette idée a permis de définir des théorèmes de filtrage [10] qui peuvent s'avérer incomparables [1] avec les algorithmes ad-hoc pour les contraintes considérées. La figure 6 illustre la complexité temporelle des propagateurs utilisant ces composants génériques.

L'innovation théorique la plus intéressante est que les propagateurs peuvent être générés par du code, sans besoin de les implémenter, à partir de la description des contraintes à l'aide de ce petit ensemble de propriétés de graphe. En pratique, un verrou nous a semblé être le calcul incrémental de ces propriétés. Une perspective peut être l'usage de techniques d'approximation.

Enfin, N. Beldiceanu et al. ont utilisé les propriétés de graphe pour générer des invariants communs à des groupes de contraintes [13]. Nous avons également utilisé cette représentation pour générer des mesures de violations de contraintes tractables dans le cas de checkers [20]. Des approches alternatives ont été proposées pour générer automatiquement des propagateurs [144, 9, 11, 141].

### 2.2 Représentation des contraintes par des automates

Cette section est relative à la représentation de contraintes dérivée des checkers de contraintes présentée dans N. Beldiceanu et al. [16, 9].

1. Et donc faire des déductions qu'un algorithme standard ne fera pas.

Ma participation à ce travail de recherche a été essentiellement ciblée sur deux aspects importants mais connexes à l'idée de la représentation que l'on doit à N. Beldiceanu et M. Carlsson : d'une part, identifier les propriétés garantissant des propagateurs GAC pour les contraintes représentées par des automates, e.g., la Berge-acyclicité ; d'autre part, traiter le cas de contraintes souples [2].

Cette représentation présente un intérêt théorique équivalent à la précédente, auquel s'ajoute un intérêt opérationnel qui est à mon avis plus important.

**Principe et propriétés.** L'originalité de l'approche est de définir un automate représentant une contrainte sous une forme indépendante des domaines, ou, en d'autres termes, qui n'est pas basée sur les combinaisons autorisées de valeurs pour la contrainte mais sur son checker. Une fois l'automate décrivant le checker défini, l'idée est de le représenter par un ensemble de contraintes. Sachant que ces contraintes peuvent être munies de propagateurs, on obtient alors un algorithme de filtrage.

Cette formulation compacte des contraintes sous forme d'automates est ainsi tout à fait différente d'un point de vue conceptuel des autres représentations sous forme d'automates que l'on trouve en PPC [149, 2, 109].

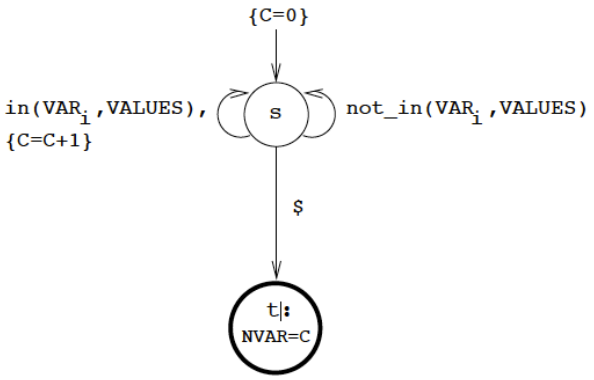


FIGURE 7 – Automate associé à la contrainte AMONG [14].

À titre d'exemple, considérons la contrainte AMONG$(N, X, T)$ qui impose que la variable $N$ soit égale au nombre de variables de l'ensemble de variables $X$ qui sont fixées avec une valeur appartenant à l'ensemble de valeurs entières $T$. La figure 7 décrit l'automate qui lui est associé.

Afin de transformer cet automate en un réseau de contraintes qui représente AMONG, une variable de signature $S_i$ est associée à chaque variable $x_i \in X$. Le domaine de chaque $s_i$ est binaire, $D(s_i) = \{0, 1\}$. La contrainte liant chaque variable de décision $x_i$ à la variable de signature $S_i$ correspondante est la suivante : $[S_i = 1 \wedge x_i \in T] \vee [S_i = 0 \wedge x_i \notin T]$. L'automate compte via des variables de transition $Q_i$ le nombre de variables de signature prenant la valeur 1.

2. Les contraintes souples seront abordées plus loin dans ce manuscrit, dans le chapitre V.

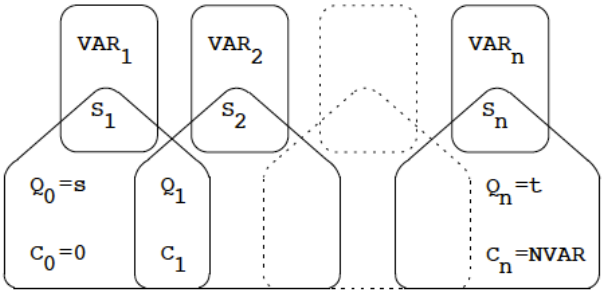


FIGURE 8 – Réseau de contraintes représentant l'automate d'AMONG [14].

Une simple contrainte d'égalité entre le compteur de la dernière variable dans $X$ et $N$ permet de représenter AMONG, comme l'illustre la figure 8.

Lorsque le réseau de contraintes ainsi défini est Berge-acyclique [23], si les contraintes primitives impliquées dans ce réseau effectuent un filtrage GAC, ce qui sera le cas dans la plupart des moteurs de résolution de contraintes, alors le réseau sera globalement consistant : au bout du cycle de propagation, les domaines seront les mêmes que si un algorithme de GAC ad-hoc avait été utilisé. Cette propriété des réseaux de contraintes Berge-acycliques a été mise en évidence par P. Jégou [71] et P. Janssen et al. [70]. Nous avons démontré qu'il est possible d'obtenir la consistance GAC dans des cas où les réseaux considérés ne sont pas Berge-acycliques, notamment grâce à d'autres types de consistance, par exemple la "pairwise consistency" [45].

Comme pour la représentation des contraintes par des graphes, la décomposition en automates permet de générer des propagateurs. La génération ne nécessite aucune implémentation spécifique si l'outil de résolution dispose des contraintes primitives utilisées pour construire l'automate correspondant à la contrainte (ce qui est le cas de Sicstus Prolog, par exemple).

**Expérimentations.** Les expérimentations réalisées sur plusieurs contraintes [9] ont démontré que l'usage d'automates Berge-acycliques n'impacte pas fondamentalement le processus de résolution. Outre l'aspect totalement générique de l'approche, une propriété intéressante est qu'il est possible de réaliser l'union d'automates, afin de générer des propagateurs raisonnant globalement sur des conjonctions de contraintes.

**Portée des travaux.** De nombreux travaux ont exploité, étendu ou bien se sont inspirés de cette représentation compacte des contraintes sous forme d'automates.

Nous pouvons notamment retenir les contraintes régissant les valeurs prises par une séquence de variables pour qu'elle définissent une chaine correspondant à un langage *context-free* [130, 128, 75, 129]. Nous pouvons également citer la contrainte SLIDE [28].

Le paradigme a été étendu à des classes plus larges d'automates et de contraintes, e.g., [12].

Des approches alternatives ont été proposées pour générer automatiquement des propagateurs [144, 141], suivant une démarche similaire à la notre.

## 3 Consistances intermédiaires et alternatives

La plupart des outils de PPC proposent des algorithmes GAC et BC pour un certain nombre de contraintes usuelles, telles que les contraintes arithmétiques, ou encore des contraintes globales [18, 32] portant sur un ensemble conséquent de variables. Le choix du propagateur a une forte influence sur le processus de résolution, au même titre que la définition d'une bonne stratégie de branchement. En général, on construit un modèle en fonction des propagateurs disponibles et des stratégies de recherche qui semblent être les plus prometteuses pour obtenir une résolution efficace.

Il existe de nombreuses contraintes pour lesquelles les propagateurs effectuent un filtrage plus faible que BC, ou bien plus fort que BC mais plus faible de GAC. Généralement, la cause est que la complexité en temps pour un filtrage GAC ou BC est NP-Difficile, mais il peut exister de bonnes raisons d'implémenter ce type de propagateurs dans d'autres cas, par exemple pour réduire le coût opérationnel du filtrage à chaque nœud de l'arbre de recherche. Il existe par exemple des problèmes de grande taille où il est tout juste raisonnable d'employer un algorithme de filtrage de complexité temporelle linéaire en le nombre de variables.

Dans la littérature, la complexité temporelle peut être considérée *au point fixe*, c'est-à-dire lorsque le propagateur a effectué toutes ses suppressions de valeurs, ou bien après une étape, dans ce cas l'algorithme peut se rappeler lui même. Ce flou complique la comparaison des propagateurs, car les conséquences opérationnelles sont importantes si la convergence au point fixe se fait en un nombre d'étapes important [97].

Dans ce contexte, les notions de BC, GAC (et RC) s'avèrent insuffisantes pour caractériser de façon formelle les algorithmes de filtrage.

Il est surprenant que peu de travaux abordent ces problèmes dans la littérature, du moins de manière générique, c'est-à-dire indépendante d'une contrainte particulière ou d'un problème particulier. Il existe certes dans l'état de l'art des caractérisations des propriétés que doivent satisfaire les propagateurs pour pouvoir être intégrées dans des outils opérationnels [145]. Notamment, la contractance et la monotonie. Ces notions ne permettent pas de classifier les propagateurs selon un ordre partiel, défini à l'aide des métriques qui sont utiles en pratique pour les comparer.

Outre les notions de puissance du filtrage, de complexité et de convergence, il peut-être pertinent de prendre en compte l'usage de structures de données coûteuses à mettre à jour, la possibilité de réagir à des événements fins, la possibilité de raisonner localement sur un sous-ensemble de variables, etc.

### 3.1 Pourquoi caractériser les propagateurs ?

Il est dommage que les moteurs actuels ne proposent aucune automatisation du choix des propagateurs, choix qui reste statique et délégué à l'utilisateur du système. Il semble pourtant difficile de demander à un développeur non expert de connaître en détail l'algorithme caché derrière chaque propagateur, condition préalable à une utilisation optimale des outils actuels.

Une première étape vers une telle automatisation serait de disposer d'outils théoriques pour comparer les propagateurs, selon des métriques qui ont du sens en pratique. Cette étape est complémentaire à une autre approche consistant à raisonner de façon probabiliste sur l'intérêt de déclencher un propagateur, dynamiquement pendant la recherche [35], approche qui n'est pas abordée ici.

Nous considérons à présent deux exemples de contraintes pour lesquelles il existe plusieurs propagateurs dans la majorité des moteurs de résolution de systèmes à contraintes : la contrainte Cumulative [18] et la contrainte AllDiff [131].

**La contrainte** Cumulative**.** L'exemple de la contrainte Cumulative (Définition 5, section 3) est intéressant, car effectuer un filtrage exhaustif des bornes des variables de début des activités est un problème NP-Difficile [3]. L'ensemble des propagateurs proposés dans la littérature effectuent donc un filtrage plus faible que BC.

Ces propagateurs sont découpés en deux grandes familles : les algorithmes topologiques [7, 92, 91] et les algorithmes énergétiques [94, 3, 146, 4, 98, 150, 76, 151].

S'il est avéré que les premiers permettent de résoudre des problèmes "simples" comportant plusieurs milliers d'activités, il est plus ou moins admis que les seconds sont plus efficaces pour résoudre de petites instances très combinatoires, ainsi que pour prouver l'optimalité dans des problèmes consistant par exemple à minimiser la date de fin d'activité la plus tardive. Néanmoins, cette affirmation n'a jamais été démontrée sans ambiguité en pratique, car la difficulté du problème rend le plus souvent la résolution extrêmement dépendante des stratégies de choix de variables et de valeurs. Ces stratégies peuvent alors tout à fait être perturbées par un filtrage un peu plus fort, notamment lorsqu'elles visent à trouver une "bonne solution" sans garantie d'optimalité.

Dans le cadre de la thèse d'Alban Derrien, nous avons montré expérimentalement [48, 49] qu'en l'absence de stratégie de recherche dédiée un simple checker énergétique peut s'avérer plus efficace que le meilleur algorithme énergétique de la littérature, Time-Table Edge-Finding (TTEF) [151]. Le but de l'expérimentation était de prouver l'optimalité pour de petits problèmes aléatoires comportant 20 activités. En utilisant un algorithme topologique associé à un checker énergétique, nous avons prouvé l'optimalité de 72% des instances. Avec le même algorithme topologique et TTEF, seulement 7% des instances ont été prouvées dans un temps raisonnable.

Cette expérience se situe pleinement dans le contexte d'un utilisateur qui n'est pas un expert. Les résultats sont contre-intuitifs car on pourrait penser qu'au contraire la puissance du filtrage compense la faiblesse de l'heuristique. Le problème dans le cas des instances que nous avons considérées

était que TTEF propose une approximation du raisonnement énergétique complet [3], qui filtre des valeurs certes mais ne réalise pas un checker énergétique complet, qui s'est avéré crucial sur ces instances. Il illustre la difficulté du choix et de la caractérisation des propagateurs associés à une contrainte.

Dans l'état de l'art, Mercier et van Hentenryck on montré que la convergence au point fixe était un critère à mettre en balance avec la puissance du filtrage réalisé, dans le cadre de propagateurs énergétiques [97]. Il peut être contre-productif de vouloir trop filtrer si la conséquence est une convergence qui a un comportement exponentiel.

**La contrainte** AllDiff **dans Choco.** La contrainte AllDiff est utile dans de nombreuses applications, notamment des problèmes d'affectation et d'ordonnancement.

**Définition 7 (AllDiff)** *Étant donné un ensemble de variables* $X = \{x_1, x_2, \ldots, x_n\}$*, l'ensemble des solutions de la contrainte* AllDiff*(X) est* $\{(v_1, v_2, \ldots, v_n), \forall i \in \{1, 2, \ldots, n\}, \forall j \in \{1, 2, \ldots, n\}, i \neq j : x_i \neq x_j\}$.

Le moteur de résolution de contraintes Choco 3.0 [39] propose plusieurs propagateurs pour la contrainte AllDiff(X), convergeant tous en une passe.

- `"GAC"` : Une implémentation de l'algorithme GAC de J.-C. Régin [131], de complexité $O(m\sqrt{|X|})$, où $|X|$ est le nombre de variables et $m$ le nombre d'arcs dans le graphe biparti liant les variables aux différentes valeurs présentes dans les domaines. Ce propagateur maintient donc un graphe et applique des algorithmes sur ce graphe (recherche de composantes fortement connexes, couplage maximal).
- `"BC"` : Une implémentation de l'algorithme de BC proposé par A. Lopez-Ortiz et al. [95], de complexité $O(n + sort)$ où *sort* est le temps nécessaire au tri des bornes des variables (selon l'écart entre la plus petite et la plus grande valeur, un tri linéaire peut être possible en utilisant un tableau). Cet algorithme est plus léger que le précédent en termes de structures de données.
- `"NEQS"` : Un propagateur simulant une clique de contraintes d'inégalités binaires, en $O(n^2)$.

Le cas de cette contrainte semble beaucoup plus simple que celui de Cumulative. L'algorithme effectuant la GAC n'est pas trop coûteux par rapport aux autres, il semble donc pertinent de l'utiliser systématiquement. En pratique, cette conclusion est plus mitigée, à cause du maintien d'un graphe notamment.

En effet, la plupart des exemples du package `samples` sont résolus plus rapidement avec un propagateur effectuant la BC qu'avec les deux autres, essentiellement parce que la contrainte AllDiff ne s'avère pas centrale dans les problèmes considérés.

| $n$ | NEQS | BC | GAC |
|---|---|---|---|
| 4 | 4 nœuds<br>0 s | 4 nœuds<br>0 s | 4 nœuds<br>0 s |
| 8 | 56 nœuds<br>0.1 s | 30 nœuds<br>0 s | 17 nœuds<br>0 s |
| 12 | $>$ 10000 nœuds<br>- | 32903 nœuds<br>6 s | 45 nœuds<br>0 s |
| 16 | $>$ 10000 nœuds<br>- | $>$ 10000 nœuds<br>- | 89 nœuds<br>0 s |
| 20 | $>$ 10000 nœuds<br>- | $>$ 10000 nœuds<br>- | 149 nœuds<br>0.1 s |
| 24 | $>$ 10000 nœuds<br>- | $>$ 10000 nœuds<br>- | 225 nœuds<br>0.2 s |
| 28 | $>$ 10000 nœuds<br>- | $>$ 10000 nœuds<br>- | 317 nœuds<br>0.7 s |

TABLE 3 – Minimisation du nombre de "breaks" pour le problème de Sport League Scheduling [133] avec Choco 3.1.0, avec différents propagateurs pour la contrainte ALLDIFF. Les temps de résolution sont indiqués en secondes.

Inversement, une implémentation en Choco 3.1.0 d'un benchmark de Sport League Scheduling illustre l'importance du propagateur effectuant la GAC dans certains problèmes. $n$ désigne le nombre d'équipes. Le but est de démontrer l'optimalité des $n-2$ "breaks" sur une demi saison, sans utiliser de méthode constructive. Les résultats sont présentés dans la table 3.1. Le même modèle [133] et les mêmes stratégies de recherche ont été utilisées, avec des propagateurs différents pour la contrainte ALLDIFF.

L'exemple de la contrainte ALLDIFF tend à démontrer qu' il demeurera sans doute toujours une part d'incertitude dans les critères de choix des propagateurs. Cette incertitude est inhérente à la sémantique des problèmes et une automatisation totale demeure pour le moment hors de portée. Mon point de vue est que cette difficulté ne doit pas dissuader la communauté de progresser sur la caractérisation formelle des propagateurs ; bien au contraire, elle souligne la pertinence du sujet. Chaque diminution de cette part d'incertitude, même mineure, peut améliorer sensiblement la diffusion de la PPC.

### 3.2 Le filtrage caractérisé par relaxation

**Principe.** Lorsque nous considérons un propagateur, le filtrage qu'il réalise consiste à supprimer des domaines de variables certaines valeurs ne pouvant pas satisfaire la contrainte à laquelle est associé le propagateur. Mon point de vue sur le sujet [113], inspiré d'autres disciplines de recherche opérationnelle, est centré sur la notion de *relaxation*.

Étant donnée une contrainte, nous appelons une relaxation une relation telle que l'ensemble des solutions de la contrainte initiale soit strictement inclus dans l'ensemble des solutions de la relaxation. Cette relaxation peut être obtenue soit en relâchant directement la contrainte (sous la

forme d'une contrainte ou d'un réseau de contraintes), soit, par exemple, en considérant que les domaines des variables comportent plus de valeurs.

Si une valeur n'appartient à aucune solution de la relaxation, alors elle peut être supprimée de son domaine. En d'autres termes, si elle n'a pas de support sur la relaxation elle n'en aura pas, a fortiori, sur la contrainte initiale.

L'idée est alors de déterminer ces relaxations en partant des algorithmes de filtrage, ou propagateurs, existant dans les moteurs de résolution ou bien proposés dans des articles de la littérature. Cette caractérisation est exacte : il est possible d'utiliser diverses métriques pour comparer les relaxations, et donc les propagateurs.

Plus précisément, nous caractérisons un propagateur par deux entités.

1. L'ensemble des valeurs que l'on cherche à filtrer : soit toutes les valeurs dans les domaines, soit les bornes.
2. La relaxation de la contrainte qui est utilisée pour tester si les valeurs que l'on cherche à filtrer on un support. Cette relaxation définit une relation dans laquelle l'ensemble des solutions de la contrainte est inclus, non nécessairement de façon stricte car GAC, par exemple, est une des caractérisations possibles.

**Définition 8 (Support-directed consistency)** *Soit une contrainte $C(X)$. Soit un ensemble de variables $Y = \{y_1, y_1, \ldots, y_n\}$ en bijection avec $X = \{x_1, x_2, \ldots, x_n\}$, de sorte que pour tout $i$ positif et inférieur ou égal à $n$, $D(x_i) \subseteq D(y_i)$. Soit $\mathcal{C} = \{C_1(Y_1), C_2(Y_2), \ldots, C_m(Y_m)\}$ un ensemble de contraintes tel que :*

- *$Y_1 \cup Y_2 \ldots Y_m \subseteq Y$*
- *et $C(Y) \Rightarrow C_1(Y_1) \wedge C_2(Y_2) \wedge \ldots \wedge C_m(Y_m)$*

*Une valeur $v \in D(x_i)$ a un $(\mathcal{C}, Y)$-**support** sur la contrainte $C(X)$ si et seulement si $\forall C_j(Y_j) \in \mathcal{C}$, soit la variable $y_i$ associée à $x_i$ n'est pas dans $Y_j$, soit $y_i \in Y_j$ et $C_j(Y_j)$ admet une solution avec $y_i = v$.*
*La contrainte $C(X)$ est $(\mathcal{C}, Y)$-**DC** ($(\mathcal{C}, Y)$-Domain Consistent) si et seulement si $\forall x_i \in X$, $\forall v \in D(x_i)$, $v$ a un $(\mathcal{C}, Y)$-support sur $C(X)$.*
*La contrainte $C(X)$ est $(\mathcal{C}, Y)$-**BC** ($(\mathcal{C}, Y)$-Bounds Consistent) si et seulement si $\forall x_i \in X$, $min(x_i)$ et $max(x_i)$ ont un $(\mathcal{C}, Y)$-support sur $C(X)$.*

L'ensemble $\mathcal{C}$ ne contient que des variables dérivées des variables de l'ensemble $X$. Le but est de caractériser des propagateurs, non pas des décompositions de contraintes ; il semble pertinent de considérer qu'un propagateur n'ajoute pas de variables au problème. Les définitions usuelles BC, GAC et RC peuvent naturellement être exprimées à l'aide de la définition 8, ainsi que de nombreuses consistances intermédiaires.

Prenons par exemple le cas d'une contrainte de somme $s = \sum_{x_i \in X} x_i$. Faire un filtrage GAC sur cette contrainte est NP-Hard [58, p. 223]. Inversement, appliquer un filtrage GAC sur $\sum_{x_i \in X} x_i \leq$

$s$ est dans $P$ [157], de même pour $\sum_{x_i \in X} x_i \geq s$. Une consistance possible pour la contrainte $s = \sum_{x_i \in X} x_i$ consiste à filtrer séparément les deux inégalités $\leq$ et $\geq$, c'est-à-dire $(\mathcal{C}, Y)$-$DC$ avec $Y = X \cup \{s\}$ et $\mathcal{C} = \{\sum_{x_i \in X} x_i \leq s, \sum_{x_i \in X} x_i \geq s\}$. Dans ce cas précis le filtrage obtenu est équivalent à BC. Ce n'est pas le cas pour d'autres contraintes, comme par exemple $s = \sum_{x_i \in X} x_i^2$ ou encore le propagateur de la contrainte CostRegular [47].

**Originalité et portée des travaux.** La définition 8 permet de définir naturellement un ordre partiel entre propagateurs d'une même contrainte. Son intérêt principal est qu'elle permet aussi de définir un ordre total, en utilisant diverses métriques [112]. Une métrique intéressante est le nombre de solutions de la relaxation, ou en d'autres termes la taille de la relation considérée pour effectuer ce filtrage. Il existe des travaux récents dans la communauté pour effectuer ce comptage, ou une approximation de ce comptage, de façon efficace [156, 37]. Une perspective consiste à embarquer dans un moteur de résolution de contraintes de tels algorithmes ou, pour des raisons d'efficacité, des approximations moins précises. Cela fournirait un critère pour sélectionner dynamiquement des propagateurs pendant la recherche d'une solution.

### 3.3 Le filtrage caractérisé par point fixe

**Principe.** Une manière alternative de caractériser un propagateur est de considérer une condition que doivent respecter les variables à son point fixe. Pour comprendre la problématique, nous allons entrer dans le détail des contraintes Cumulative et FlexC présentées précédemment.

Nous étudions tout d'abord la condition d'échec du propagateur Time-Table (TT) pour Cumulative, principe utilisé par exemple par l'algorithme de filtrage Dynamic Sweep [92]. Ce propagateur est basé sur la notion de partie obligatoire [79]. La partie obligatoire d'une activité $a$ est l'intervalle de temps qu'elle couvrira nécessairement, quel que soit sa position. Cet intervalle est $[\overline{s_a}, \underline{e_a}[$ (vide si $\overline{s_a} \geq \underline{e_a}$). Le *profil* est la hauteur cumulée des hauteurs des activités ayant une partie obligatoire, en chaque point de temps. Elle ne doit pas dépasser la capacité. Étant donnée Cumulative$(\mathcal{A}, C)$, où $C$ est la capacité entière, on a la condition suivante [8].

**Proposition 1** *Si*

$$\exists t \in \mathbb{N}, (\sum_{a \in \mathcal{A}, t \in [\overline{s_a}, \underline{e_a}[} h_a) > C$$

*alors* Cumulative$(\mathcal{A}, C)$ *n'admet pas de solution.*

A. Letort et al.'s (Property 1 dans [92]) ont explicité la condition au point fixe de leur algorithme TT *sweep_min* sur les valeurs minimales des domaines de débuts d'activités.

**Propriété 1** *Étant donnée* Cumulative$(\mathcal{A}, C)$, sweep_min *assure que*

$$\forall b \in \mathcal{A}, \forall t \in [\underline{s_b}, \underline{e_b}[, h_b + \sum_{a \in \mathcal{A} \setminus \{b\}, t \in [\overline{s_a}, \underline{e_a}[} h_a \leq C$$

Nous pouvons noter que cette définition caractérise le filtrage effectué, à partir de la condition d'échec du propagateur. Nous avons proposé d'étendre cet idée au cas de FlexC, afin de caractériser un propagateur TT pour une contrainte différente de Cumulative.

Pour ce faire, la première étape est de fournir la condition d'échec du propagateur TT de FlexC. Il est nécessaire d'ajouter à la hauteur cumulée des activités la hauteur nécessaire à prise en compte de la robustesse. Nous avons introduit la notion de partie obligatoire robuste, $\mathcal{K}$-compulsory part. Étant donnée la partie obligatoire d'une activité, la $\mathcal{K}$-compulsory part est le sous intervalle correspondant à la partie obligatoire calculée avec le slack $k_a$ en plus de la durée, à laquelle on a retiré la partie obligatoire classique de l'activité.

**Définition 9 ($\mathcal{K}$-compulsory part)** *Soit $a \in \mathcal{A}$ une activité de slack $k_a \in \mathcal{K}$. La* $\mathcal{K}$-compulsory part *de $a$, notée $KCP_a$, est l'intervalle* $[\max(\overline{s_a}, \underline{e_a}), \underline{e_a} + k_a]$.

La condition d'échec du propagateur TT de FlexC doit intégrer cette partie obligatoire robuste. À un instant $t$, nous considérons la hauteur maximale parmi les activités ayant une $\mathcal{K}$-compulsory part.

**Proposition 2** *Si il existe un instant $\exists t \in \mathbb{N}$ tel que*

$$\left(\sum_{a \in \mathcal{A}, t \in [\overline{s_a}, \underline{e_a}[} h_a\right) + \left(\max_{a \in \mathcal{A}, t \in KCP_a} h_a\right) > C$$

*alors* FlexC$(\mathcal{A}, C, \mathcal{K})$ *n'a pas de solution.*

La condition au point fixe résultant de cette proposition s'avère être un peu plus complexe que dans le cas de Cumulative. Elle permet néanmoins d'obtenir une caractérisation précise d'un propagateur TT pour une nouvelle contrainte, FlexC.

**Propriété 2 (FlexC (bornes inférieures))** *Soit* FlexC$(\mathcal{A}, C, \mathcal{K})$. *Un propagateur TT doit assurer pour toute activité $b \in \mathcal{A}$ :*

$$\forall t \in [\underline{s_b}, \underline{e_b}[, \left(h_b + \sum_{\substack{a \in \mathcal{A} \setminus \{b\} \\ t \in [\overline{s_a}, \underline{e_a}[}} h_a\right) + \left(\max_{\substack{a \in \mathcal{A}, \\ t \in KCP_a}} h_a\right) \leq C \quad (1)$$

$$\wedge \forall t \in [\underline{e_b}, \underline{e_b} + k_b[, \left(\sum_{\substack{a \in \mathcal{A} \\ t \in [\overline{s_a}, \underline{e_a}[}} h_a\right) + h_b \leq C \quad (2)$$

Le cas des bornes supérieures des domaines de variables de fin d'activité est similaire.

**Originalité et portée des travaux.** Sachant que pour certains propagateurs il peut s'avérer complexe d'extraire la relaxation comme cela est décrit dans la section précédente, ce type de caractérisation constitue une alternative valable. Néanmoins, son défaut est qu'elle ne permet pas directement de disposer de métriques pour comparer des propagateurs, sauf si les conditions au point fixe sont proches dans leur écriture.

### 3.4 Consistances fortes dans les moteurs événementiels

La plupart des solveurs de contraintes sont basés sur un schéma de propagation type AC-5 [66]. Ils sont communément appelés "solveurs événementiels". Dans un tel schéma, chaque propagateur est appelé lorsque des événements surviennent sur les domaines des variables impliquées dans chaque contrainte. À un nœud donné de l'arbre de recherche, le filtrage est réalisé à l'intérieur de chaque contrainte. Une contrainte reçoit les événements relatifs à ses variables et réalise un filtrage, qui déclenche de nouveaux événements. Le point fixe est obtenu après un cycle de propagation des événements impliquant toutes les contraintes, lorsque plus aucun nouvel événement n'est déclenché. Dans un tel contexte, la consistance d'arc généralisée (GAC) est le niveau de consistance locale le plus élevé. Il correspond au cas où tous les propagateurs sont complets.

Il existe pourtant, dans la littérature, des consistances locales plus fortes que GAC [46, 30]. Elles nécessitent la prise en compte de plusieurs contraintes simultanément pour être appliquées. On considère parfois que ces techniques ne peuvent pas être intégrées telles qu'elles aux outils de PPC, parmi lesquels, notamment, les moteurs de résolution événementiels. Ces outils ne proposent généralement pas ces consistances plus fortes, qui, en conséquence, sont rarement utilisées pour résoudre des problèmes industriels.

Une des raisons est que ces méthodes sont nativement destinées à être utilisées avec des contraintes de table, définies par la liste des combinaisons autorisées ou bien interdites. Dans le cadre de l'étude post-doctorale de Julien Vion, nous avons démontré [152] que les consistances fortes sont exclues à tort des outils de PPC.

**Principe.** Nous avons proposé un paradigme générique pour ajouter facilement de telles consistances aux solveurs de contraintes. Notre idée est de définir une contrainte globale [32, 14], qui encapsule un sous-ensemble du réseau de contraintes initial. Le niveau de consistance souhaité est alors appliqué sur ce sous-ensemble de contraintes. On notera que, généralement, une contrainte globale représente un sous-problème ayant une sémantique bien fixée. Ce n'est pas le cas dans notre approche. La contrainte globale est ici utilisée pour appliquer une technique de propagation particulière sur un sous-ensemble de contraintes. Cette idée est similaire à celle utilisée pour la résolution de problèmes sur-contraints [5, 136, 137].

En outre, notre approche permet de combiner des consistances fortes telles que Max-RPC ou Light Max-RPC (voir [152]) avec BC, GAC ou RC, pour résoudre un problème modélisé par un réseau de contraintes.

**Expérimentations.** Le tableau 4 illustre l'intérêt d'une telle combinaison sur des instances aléatoires. L'implémentation a été réalisée en Choco 2.0. Un problème aléatoire est caractérisé par un quadruplet $(n, d, \gamma, t)$, dont les éléments représentent respectivement le nombre de variables, le nombre de valeurs, la densité (proportion de contraintes dans le graphe par rapport au nombre maximum possible de contraintes $\gamma = e/\binom{n}{2}$) et la dureté des contraintes (proportion d'instancia-

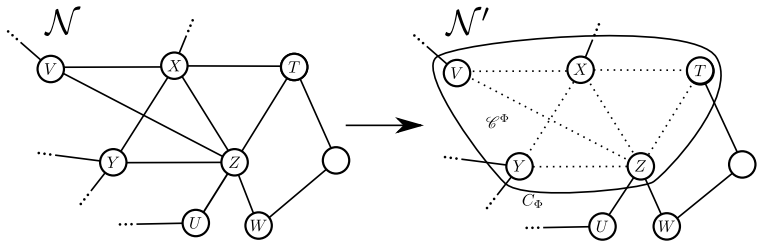


FIGURE 9 – Une contrainte globale $C_\Phi$, utilisée pour appliquer une consistance forte $\Phi$ sur un sous-ensemble de contraintes $\mathscr{C}^\Phi$. $\mathcal{N}'$ est le réseau de contraintes obtenu en remplaçant $\mathscr{C}^\Phi$ par la nouvelle contrainte globale.

| | | AC | MaxRPC | Light | AC+MaxRPC | AC+Light |
|---|---|---|---|---|---|---|
| $(35, 17, 44\%, 31\%)$ | *cpu (s)* | **6.1** | 25.6 | 11.6 | pas | pas |
| | *nœuds* | 21.4k | 5.8k | 8.6k | applicable | applicable |
| $(105, 20, 5\%, 65\%)$ | *cpu (s)* | 20.0 | 19.4 | **16.9** | pas | pas |
| | *nœuds* | 38.4 k | 20.4 k | 19.8 k | applicable | applicable |
| $(35, 17, 44\%, 31\%)$ | *cpu (s)* | 96.8 | 167.2 | 103.2 | 90.1 | **85.1** |
| $+(105, 20, 5\%, 65\%)$ | *nœuds* | 200.9k | 98.7k | 107.2k | 167.8k | 173.4k |
| $(110, 20, 5\%, 64\%)$ | *cpu (s)* | 73.0 | 60.7 | **54.7** | pas | pas |
| | *nœuds* | 126.3k | 54.6k | 56.6k | applicable | applicable |
| $(35, 17, 44\%, 31\%)$ | *cpu (s)* | 408.0 | 349.0 | 272.6 | 284.1 | **259.1** |
| $+(110, 20, 5\%, 64\%)$ | *nœuds* | 773.0k | 252.6k | 272.6k | 308.7k | 316.5k |

TABLE 4 – Combinaison de deux niveaux de consistance dans un même modèle.

tions interdites pour chaque contrainte). La première ligne correspond au résultat médian sur 50 instances de problèmes aléatoires $(35, 17, 44\%, 31\%)$ forcés à être satisfiables. A l'inverse, les "seeds" de $(105, 20, 5\%, 65\%)$ sont choisis de telle sorte que toutes les instances soient insatisfiables. Le premier problème est mieux résolu en utilisant AC-$3^{rm}$, le deuxième est mieux résolu avec Max-RPC. La troisième ligne décrit les résultat d'un problème qui concatène les deux précédents, en les liant par une unique contrainte additionnelle. Sur les deux dernières colonnes, l'AC est maintenue sur les parties denses du graphe de contraintes, et Max-RPC sur le reste du réseau.

### 3.5 Rendre la BC aussi efficace que la GAC

En collaboration avec C. Bessière et B. Zanuttini, nous avons étudié les conditions de convexité faisant que les consistances BC et GAC sont équivalentes [31]. Sachant que ces conditions dépendent de l'ordre des valeurs dans les domaines, nous avons proposé et étudié les algorithmes de ré-ordonnancement des valeurs dans les domaines permettant d'atteindre ces propriétés. Ce travail est essentiellement théorique mais une perspective intéressante est de concevoir des propagateurs dédiés, pour des contraintes spécifiques, qui exploitent l'idée de ré-ordonner les domaines avant d'effectuer un traitement, pour essayer de tendre vers ces propriétés. Sans aller jusqu'à la GAC, on peut alors raisonnablement espérer améliorer le pouvoir de suppression d'un algorithme ad-hoc réalisant la BC, par exemple.

## 4 Perspectives

Représenter de façon générique des contraintes, les décomposer sous forme de contraintes, de graphes ou d'automates, générer automatiquement le code des propagateurs, ré-ordonner les valeurs pour améliorer le filtrage, classifier les propagateurs pour les appeler dynamiquement sans solliciter l'expertise de l'utilisateur d'un système, ou encore combiner intelligemment différentes techniques de filtrage, constituent autant d'approches visant à augmenter la déclarativité de la PPC. Ces voies de recherche sont cruciales pour l'avenir de la discipline.

De mon point de vue, il s'agit encore d'une recherche scientifique en amont, car le pas à franchir pour concevoir des moteurs de résolution autonomes, ne nécessitant pas d'expertise pour être utilisés, est encore très important. Ce fait souligne et renforce l'importance des contributions sur le sujet, particulièrement les contributions théoriques.

Un point important et d'actualité concerne le passage à l'échelle. Dans certains contextes d'utilisation, par exemple le *cloud computing*, il est important de traiter des problèmes de grande taille, ayant un nombre de variables se comptant parfois en millions. La PPC a un rôle à jouer car son pouvoir de modélisation est intéressant même dans le cadre de très gros problèmes. Un exemple parmi d'autres est la réduction de la consommation énergétique obtenue en allouant de manière intelligente des tâches à des processeurs. Nous avons très récemment caractérisé les contraintes "auto-décomposables" afin de proposer un paradigme permettant d'utiliser les algorithmes de filtrage de façon paresseuse, en ne ciblant que certains sous-ensembles de variables [52]. Un autre sujet pouvant étendre le champ des applications de la PPC concerne le traitement efficace de réseaux de contraintes quantifiés [36] de taille importante.

Concernant les aspects opérationnels, ce thème me semble en outre plus ou moins indissociable de la nécessité d'hybrider la PPC avec d'autres techniques. Si l'hybridation avec des techniques de Recherche Opérationnelle constitue un thème de recherche largement exploité en pratique, l'hybridation avec des paradigmes issus de l'Intelligence Artificielle demeure globalement plus théorique. La mise en pratique de ces techniques, par exemple dans le domaine de l'apprentissage, devrait permettre d'attaquer avec le formalisme rigoureusement reproductible de la PPC de nouvelles classes d'applications dans le futur, par exemple l'analyse de données de très grande taille.

# V Problèmes sur-contraints

*De nombreux problèmes combinatoires sont soumis à des contraintes ou à des préférences contradictoires et n'admettent pas de solution satisfaisant l'ensemble des contraintes. Ce cas est très fréquent en pratique. Un moteur de résolution de PPC répondra "*`no solution`*" si on tente de résoudre un tel problème bien que dans le monde réel des solutions soient mises en place. Les applications pratiques sont souvent traitées de façon partielle et empirique, en supprimant ou en modifiant à la main certaines contraintes du modèle. Une autre approche consiste à utiliser la puissance de calcul du système pour trouver un compromis qui soit le meilleur possible, à partir de l'ensemble des contraintes initialement posées. Le problème ainsi défini est un problème d'optimisation. Ce chapitre présente les approches existantes et une sélection de mes contributions sur ce sujet, dans le cadre de la PPC.*
*Traiter les problèmes sur-contraints fait partie intégrante du thème de ce manuscrit. En effet, fournir des solutions violant certaines contraintes implique de disposer d'outils de modélisation très pointus, car ces solutions ne seront d'aucune aide si elles sont trop désincarnées des situations pratiques. D'un autre côté, les problèmes d'optimisation induits peuvent s'avérer extrêmement complexes à résoudre.*

**Note :** *Sachant qu'une partie significative de ces avancées est issue de ma thèse ou fait suite à ma thèse, j'ai fait le choix d'une description courte. J'invite les lecteurs intéressés par plus de détails techniques à consulter ma thèse* [111] *ainsi que les publications mentionnées ci-dessous.*

*Collaborations : Pr. Nicolas Beldiceanu, Dr. Hab. Christian Bessière, Dr. Alexis De Clercq, Dr. Emmanuel Poder, Dr. Jean-François Puget, Pr. Jean-Charles Régin.*
*Publications majeures : CP2000, IEEE-ICTAI2000, CP2001 (2), CP2002, CPAIOR2004, CPAIOR2008, Annals of OR, CPAIOR2011, CP2011.*

## 1 Contexte et Problématique

Lorsqu'on se trouve confronté à un problème sur-contraint, le but est de fournir un compromis acceptable en pratique. Pour obtenir ce compromis, on doit considérer une relaxation du problème : les solutions peuvent violer certaines contraintes. Le but est de limiter les violations, afin d'obtenir une solution aussi proche que possible de la solution idéale impossible à atteindre.

Ce compromis doit généralement respecter des contraintes métier qui portent sur les violations elles-même. Par exemple, dans un problème de gestion de personnel on peut souhaiter répartir équitablement les violations des préférences horaires exprimées par chaque employé. En outre, des contraintes dites *dures* doivent impérativement être satisfaites. Par exemple, un employé ne pourra pas être affecté à deux endroits différents au même horaire.

Dans ce chapitre, le terme *relaxation de contrainte* signifiera que l'on augmente le nombre de solutions d'une contrainte. Une relaxation du problème consiste alors à relâcher une ou plusieurs contraintes utilisées pour le modéliser.

Une première approche, meilleure que celle qui consiste à effectuer une relaxation empirique et désordonnée du problème jusqu'à obtenir des instances satisfiables, est d'exploiter un mécanisme d'explications [72, 74, 73]. Une explication est un ensemble de décisions (contraintes) qui peut justifier par exemple la suppression d'une valeur, et donc le fait qu'un domaine soit vidé. S'appuyer sur des explications peut s'avérer très utile pour déterminer quelles sont les contraintes à modifier pour obtenir une solution qui constitue un compromis acceptable en pratique.

Une deuxième approche, celle qui nous intéresse ici, consiste à utiliser la puissance de calcul du moteur de résolution pour trouver le meilleur compromis possible. Une notion simple à la base de cette approche est la notion de *quantification* d'une violation de contrainte. Dans certain cas, plutôt minoritaires, celle-ci est naturelle, par exemple la contrainte $x < y$ est davantage violée si $x = 100$ et $y = 2$ que si $x = 3$ et $y = 2$. Si la quantification est binaire, c'est-à-dire, réduite aux états "contrainte satisfaite" et "contrainte violée", on obtient le problème Max-CSP, qui consiste à minimiser le nombre de violations de contraintes dans les solutions. Le plus souvent, les algorithmes de résolution de Max-CSP peuvent être étendus quasi directement à des problèmes où la quantification est plus fine, ce qui justifie que Max-CSP ait été un des problèmes les plus étudiés dans la littérature. En PPC, l'idée de traiter un problème sur-contraint sous la forme d'un problème d'optimisation a été introduite par E. C. Freuder [55] et se décline en deux familles de méthodes.

- La première famille consiste à étendre la PPC pour définir un cadre formel où les violations de contraintes sont quantifiées à l'aide de *valeurs* de coût. Sachant que l'on manipule des valeurs, on peut alors considérer que les contraintes sont des fonctions qui retournent une valeur, d'où le nom (récent) de CSP fonctionnels, anciennement CSP valués ou semi-anneau [34, 83].
- La deuxième famille [121], introduite dans le cadre de ma thèse encadrée par Christian Bessière et Jean-Charles Régin [111], s'attaque aux problèmes où il est nécessaire ou pertinent de quantifier les violations de contraintes par des variables.

**CSP fonctionnels [34, 83].** Le principe est d'associer à chaque contrainte $C$ une valuation dépendante des valeurs affectées aux variables de $C$. Cette valuation est prise dans un ensemble $E$ ordonné. Un critère d'optimisation est alors défini sur l'ensemble des valuations. Lorsque plusieurs contraintes sont violées, la combinaison des valuations est effectuée selon une loi de composition interne $\oplus$. Etant donnés $E, \oplus$ et une relation d'ordre $\succ$, on définit :

**Définition 10 : structure de valuation**
*Une structure de valuation est un triplet* $(E, \succ, \oplus)$ *tel que :*

- *E soit un ensemble totalement ordonné par* $\succ$*, muni d'un élément minimum noté* $\bot$ *et un élément maximum noté* $\top$*.*
- *E soit muni d'une loi de composition interne commutative et associative notée* $\oplus$ *qui vérifie :*
  - $\forall a, b, c \in E$ *tels que* $b \succeq c$ *on* $a : (a \oplus b) \succeq (a \oplus c)$
  - *élément neutre :* $\forall a \in E, a \oplus \bot = a$
  - *élément absorbant :* $\forall a \in E, a \oplus \top = \top$

**Définition 11 : CSP valué**

*Un CSP valué* $VCSP = \{\mathcal{X}, \mathcal{D}, \mathcal{C}, S, \varphi\}$ *est défini par :*

- *un réseau de contraintes* $\mathcal{R} = \{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$*,*
- *une structure de valuation* $S = (E, \succ, \oplus)$*,*
- *une application* $\varphi$ *de* $\mathcal{C}$ *dans* $E$ *associant une valuation à chaque contrainte.*

Dans la littérature, d'autres paradigmes ont été définis. Ils correspondent à des cas particuliers de CSP fonctionnels. On peut notamment citer les CSP pondérés [56], les CSP possibilistes [140] , les CSP lexicographiques [53] et les CSP flous [51]. Le tableau ci-après [111] rappelle les paramètres correspondant à chaque classe de problème.

| Classe | $e \in E$ | $\oplus$ | $\bot$ | $\top$ | $\succ$ |
|---|---|---|---|---|---|
| CSP pondérés | $[0, n]$ | $+$ | 0 | $\infty$ | $>$ |
| CSP possibilistes | $[0, 1]$ | $max$ | 0 | 1 | $>$ |
| CSP flous | $[0, 1]$ | $min$ | 1 | 0 | $<$ |
| CSP lexicographiques | $[0, 1]^* \cup \top$ | $\cup$ | $\emptyset$ | $\top$ | lexicographique |

**Approche utilisant des variables [121, 111].** Nous avons démontré que l'ensemble de ces paradigmes peuvent être adaptés au cadre standard de la PPC sans augmentation de la complexité spatiale. Ce point de vue est générique. Il permet d'utiliser le moteur de résolution de contraintes de son choix pour modéliser n'importe quel problème à sur-contraint.

Succinctement, nous pouvons décrire le principe ainsi : soit un réseau de contraintes $\mathcal{R} = \{X, D, \mathcal{C}\}$. Nous consédérons de façon séparée les contraintes dures $\mathcal{C}_d \subseteq \mathcal{C}$ et les contraintes souples du problème $\mathcal{C}_m \subseteq \mathcal{C}$. Nous pouvons alors associer à chaque contrainte souple $C_{m_i} \in \mathcal{C}_m$, une variable de coût $cost_i$, telle que si $C_{m_i}$ est satisfaite alors $cost_i = 0$, sinon $cost_i > 0$. Pour exprimer une telle conditionnelle il convient de remplacer chaque contrainte souple $C_{m_i} \in \mathcal{C}_m$ par une contrainte exprimant la conditionnelle.

La manière dont cette conditionnelle est représentée, par une disjonction $(C_{m_i} \wedge cost_i = 0) \vee (\neg C_{mi} \wedge cost_i > 0)$ ou bien par une nouvelle contrainte $C'_{m_i}$ impliquant en plus des variables de décision la variable de violation $cost_i$, est un problème relatif à l'implémentation. Cet aspect est dépendant des outils employés.

La représentation sous la forme d'une disjonction induit l'idée élégante de n'utiliser que des contraintes dures, sans ré-écriture de contraintes spécifiques [134]. Elle implique que l'outil propose

des implémentations des négations des contraintes. En étudiant les contraintes dans le détail, nous pouvons nous rendre compte que le traitement de propagation relatif au fait d'imposer une violation ($cost_i > 0$) s'avère être NP-Difficile dans plusieurs cas de contraintes usuelles, pour nombre de définitions de la quantification des violations. Mon point de vue est qu'il est préférable de définir directement les contraintes avec une variable de coût. D'une part ce principe est plus simple. D'autre part, munir par défaut les contraintes d'un mécanisme de coût entier (par extension d'une simple réification booléenne) fournit directement à l'utilisateur la possibilité de résoudre des problèmes à l'aide de techniques de recherche locale, en utilisant les checkers.

Ce principe est par exemple utilisé dans le système COMET [147], premier système efficace de recherche locale basée sur des modèles à contraintes. L'efficacité du système est due, entre autres, à la possibilité de réaliser très rapidement un très grand nombre de mouvements, à la vitesse d'exploration du voisinage, et donc à un calcul incrémental performant des coûts quantifiant les violations. Plus que le principe lui même, ces mécanismes efficaces constituent des contributions majeures au domaine de la PPC.

## 2 Contributions et portée des travaux

Mon manuscrit de thèse [111] et les articles qui ont suivi fournissent une description plus détaillée des contributions listées ci-dessous.

### 2.1 Un nouveau modèle

Avec J.-C. Régin et C. Bessière [121], nous avons proposé l'approche orientée variables décrite précédemment.

Dans ce cadre, une contribution importante a été de montrer que les critères de qualité des solutions violant des contraintes n'étaient pas facilement exprimables à l'aide d'une fonction. Notre modèle a un pouvoir expressif permettant d'exprimer des concepts plus fins, comme par exemple une répartition topologique équilibrée des violations de contraintes. Il suffit d'utiliser des contraintes portant sur les variables qui représentent les violations, par exemple des contraintes de cardinalité.

Le concept est directement extensible au cadre plus général des problèmes d'optimisation. C'est une force de la PPC et de nombreuses contraintes permettant d'exprimer des sémantiques non fonctionnelles sur des coûts ont été proposées par la suite, e.g., [110, 138, 118, 119, 112, 103]. Certaines de ces contraintes seront décrites dans le prochain chapitre. L'article de P. Schaus et al. [139], par exemple, illustre l'intérêt applicatif de ces contraintes.

De plus, ce cadre permet une back-propagation du critère d'optimisation et des événements sur les variables de violation, parfois très utile pour la résolution, ce qui n'est pas possible dans le cadre des CSP fonctionnels. Nous avons montré avec E. Poder [116, 117] qu'il existe des classes de problèmes où il est indispensable de disposer de tel mécanismes de propagation. L'approche orientée

variable semble donc pertinente lorsque des contraintes métier sont définies sur les variables exprimant les violations de contraintes. La table 5 suivante est extraite de cette expérimentation [116].

| Instance | Excés de capacité | Valuation-based Model | Variable-based Model |
|---|---|---|---|
| 1 | 2 | $> 60$ s | **211 (0.34 s)** |
| 2 | 0 | $> 60$ s | **178 (0.08 s)** |
| 3 | 1 | $> 60$ s | **200 (0.12 s)** |
| 4 | 15 | $> 60$ s | **27 (0.04 s)** |
| 5 | 12 | $> 60$ s | **546 (2 s)** |
| 6 | 3 | $> 60$ s | **1875 (6 s)** |
| 7 | 6 | 2240 (11.4 s) | **160 (0.12 s)** |
| 8 | 9 | $> 60$ s | **79 (0.06 s)** |

TABLE 5 – Nombre de nœuds pour prouver l'optimalité sur des problèmes cumulatifs sur-contraints avec des contrainte annexes (répartition homogène des dépassements de ressource), 15 activités, 21 intervalles de temps, avec un dépassement maximum autorisé doublant la capacité initiale.

À l'opposé, les CSP fonctionnels n'utilisant pas de variable de coût, ils n'imposent pas un comportement monotone de ce coût durant la recherche. Si j'ai prouvé dans ma thèse que tous les algorithmes de résolution génériques de Max-CSP basés sur des opérations de projections peuvent être adaptés à un modèle utilisant des variables, sans aucun surcoût théorique ou pratique, cela n'est plus vrai pour les algorithmes qui exploitent des mécanismes d'extension des coûts vers les contraintes [83].

Les deux approches sont donc complémentaires, selon le type de problèmes, notamment la présence ou non de contraintes métier sur les coûts mesurant la violation de certaines contraintes, car leur propagation (imposant l'usage de variables) peut être essentielle au processus de résolution.

### 2.2 Une contrainte pour Max-CSP

Avec J.-C. Régin, C. Bessière et J.-F. Puget, nous avons proposé d'encapsuler dans une unique contrainte globale le problème Max-CSP, consistant à minimiser le nombre de violations dans un réseau de contraintes [136, 137]. Cette approche générique a été reprise ensuite pour représenter des CSP valués [88].

L'idée avait été proposée précédemment par P. Baptiste et al. [5] dans un cadre plus spécifique, les problèmes d'ordonnancement sur-contraints.

### 2.3 Généralisation des algorithmes de résolution

À partir d'une synthèse unifiée des algorithmes de résolution de l'état de l'art pour Max-CSP que j'ai réalisée pendant mon stage de master, avec J.-C. Régin, C. Bessière et J.-F. Puget nous avons généralisé le meilleur algorithme de l'époque, PFC-MRDAC [82], restreint à des contraintes

binaires (deux variables), au cas général [136, 111]. L'idée de la généralisation est simple : remplacer le graphe orienté des contraintes binaires pris en compte dans PFC-MRDAC par une partition des contraintes où chaque classe est identifiée par une variable du problème. Ces algorithmes sont directement extensibles au cas de contraintes pondérées.

Nos algorithmes ont été reformulés dans le cadre des CSP fonctionnels par des auteurs qui participaient au projet de recherche européen ECSPLAIN finançant ma thèse [99].

J.C.-Régin a proposé l'idée de les généraliser au cas du traitement d'une somme avec contraintes annexes [135]. Je pense que cette idée est pertinente et mériterait d'être approfondie, car la propagation des contraintes représentant un critère d'optimisation est relativement faible en PPC, ce qui interdit le plus souvent d'envisager l'utilisation d'une méthode de recherche exacte. Cette idée peut être vue comme une spécialisation des consistances fortes abordées précédemment au cas des contraintes de somme.

### 2.4 Un nouvel algorithme pour Max-CSP

Avec J.-C. Régin, C. Bessière et J.-F. Puget, nous avons proposé un algorithme basé sur les ensemble de contraintes en conflit pour résoudre Max-CSP [137].

Un ensemble de contrainte en conflit est détecté en propageant successivement les contraintes jusqu'à ce que le domaine d'une variable soit vidé. Si les contraintes sont munies de propagateur GAC, le principe ne dépend pas de l'ordre dans lequel ces propagations sont effectuées. Il est possible de générer avec un nombre linéaire d'étapes des ensemble en conflits minimaux au sens de l'inclusion [69] (ou encore logarithmique en procédant par dichotomie). Sachant que chaque ensemble en conflit engendrera au moins une violation, par une génération d'ensemble disjoints on peut obtenir une borne inférieure du nombre de contraintes violées. Cet algorithme est incomparable avec ceux de l'état de l'art concernant la détection en amont des violations dans un réseau de contraintes, en cours de recherche [111].

### 2.5 Technique de filtrage aux bornes

J'ai conçu un algorithme de filtrage aux bornes pour Max-CSP [123]. Cet algorithme est directement extensible au cas de contraintes pondérées. Son principe consiste à propager indépendamment chaque contrainte sur le domaine d'une variable, puis à balayer les bornes domaines propagés ainsi obtenus de la plus petite à la plus grande valeur. Lors du balayage, l'algorithme maintient incrémentalement un compteur du nombre de contraintes satisfaites à chaque point de temps correspondant à la borne inférieure ou supérieure d'un domaine. En répétant le processus sur chaque variable, via un processus de partitionnement similaire à notre généralisation de PFC-MRDAC il est possible d'obtenir un algorithme de filtrage pour Max-CSP indépendant de la taille des domaines. Cet algorithme a été amélioré par la suite [158], dans le cadre des CSP fonctionnels, et son principe exploité dans d'autres contextes, e.g., pour propager la contrainte SoftAllEqual [63].

## 2.6 Problèmes d'ordonnancement sur-contraints

En collaboration avec E. Poder puis dans le cadre de la thèse d'A. De Clercq nous avons proposé une contrainte globale pour les CuSP sur-contraints, SoftCumulative [116, 117, 40]. Cette contrainte permet d'exprimer de manière propre des problèmes cumulatifs avec dépassement de ressource. Nous avons adapté les algorithmes Time-Table [8, 92] et Edge-Finding [150].

En considérant la contrainte de façon isolée, cette approche par une contrainte dédiée est équivalente à une modélisation alternative où l'on ajoute des activités fictives, fixées, pour limiter l'espace disponible pour d'autres activités dans certains intervalles de temps.

Cette modélisation alternative n'est cependant pas adaptée à tous les contextes car elle mélange des concepts distincts (coûts de violation, activités) et ce mélange peut s'avérer incompatible avec certains aspects du modèle.

De plus, il n'est pas possible de définir tous les principes de propagation embarqués dans notre contrainte.

Enfin, il arrive que l'utilisateur souhaite aussi modéliser une consommation minimale de ressource dans certains intervalles de temps. Cela peut se faire par des activités de hauteurs négatives en imposant que la consommation en tout point de temps demeure positive. Sans au moins une contrainte dédiée (pour les dépassements ou pour la consommation minimale), le phénomène d'aller retour entre au moins deux Cumulative peut s'avérer pénalisant pour l'efficacité du processus de résolution.

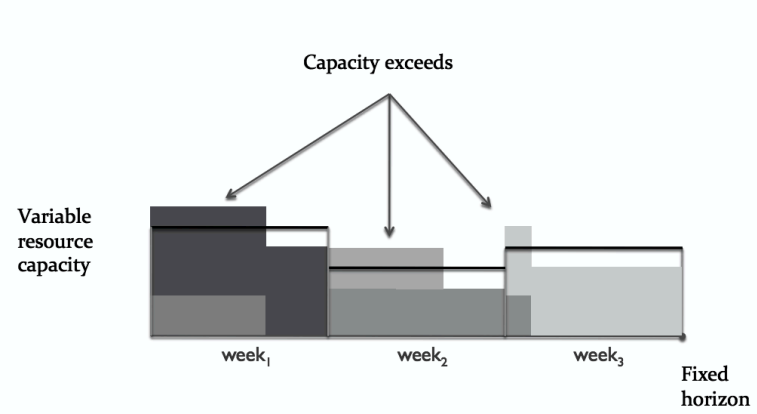


Figure 10 – Illustration d'un problème modélisé par la contrainte SoftCumulative.

## 2.7 Contraintes globales souples

J'ai proposé le concept de *contrainte globale souple*, indépendant des contraintes globales représentant un problème entier tel que Max-CSP ou CuSP. L'idée d'exploiter la structure et la sémantique des contraintes ayant un coût de violation [122], comme cela avait été fait avec succès

dans un cadre de satisfaction, par exemple avec la contrainte AllDiff et son propagateur [131], abordée dans le chapitre IV. Ces contraintes sont aussi appelées *contraintes globales souples*.

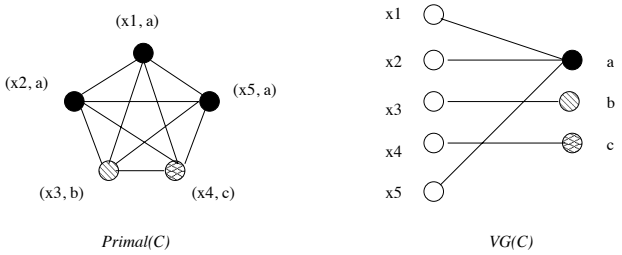


Figure 11 – Illustration de la mesure de violation issue de la décomposition de la contrainte AllDiff en une clique d'inégalités binaires [122]. $cost = (3 * 2)/2 + 0 + 0 = 3$.

Définir ce type de contraintes implique de définir des *mesures de violation génériques*, car mesurer la violation de façon correcte et pertinente n'est pas toujours facile, particulièrement pour les contraintes globales.

Par exemple, dans le cadre de la contrainte AllDiff, nous avons défini une mesure basée sur la distance de Hamming, identifiée comme pertinente dans des applications réelles par J.-F. Puget.

Nous avons étudié une mesure de violation basée sur la décomposition en contraintes primitives [122], que j'ai imaginée pour obtenir une granularité plus fine que celle de la mesure basée sur la distance de Hamming. Cette mesure de violation est illustrée par la figure 11. Elle a été reprise avec un point de vue plus général dans [96].

Nous avons proposé par la suite avec N. Beldiceanu d'autres mesures de violation, dont une basée sur les propriétés de graphes, qui garantit un calcul polynomial pour les checkers de toutes les contraintes (ce qui n'est pas le cas des autres définitions) [20].

Nous avons proposé de nouveaux propagateurs pour deux mesures de violations dans le cas de AllDiff [122], ainsi que des techniques de propagation dédiées aux contraintes représentées par des automates [9].

Ces travaux ont eu une portée importante car de nombreuses contraintes ont été transformées en contraintes globales souples, avec diverses mesures, plus ou moins génériques [148, 68, 155, 27, 100, 64, 77, 116, 87, 90, 63, 116, 117, 40, 38]. Cette liste de références n'a pas prétention à être exhaustive.

Les systèmes COMET [147], EasyLocal++ [59] et OscaR (https ://bitbucket.org/oscarlib/oscar/wiki/Home), par exemple, exploitent des mesures de violation sémantiques et génériques pour les contraintes.

## 3 Perspectives

Les problèmes sur-contraints et les outils de recherche locale basés sur des modèles à contrainte ont une grande importance en pratique et il existe de nombreux champs d'investigation encore non explorés. Les trois sujets suivants me semblent particulièrement pertinents en PPC.

1. Proposer de nouvelles techniques hybridant les paradigmes basés sur les explications et les techniques visant à représenter un problème sur-contraint comme un problème d'optimisation. La séparation entre ces deux approches me semble essentiellement historique.
2. Établir une classification des contraintes "métier" sur les variables représentant des degrés de violation, par familles d'applications ou familles de concepts. Certaines de ces contraintes doivent être exprimées telles qu'elles, d'autres via des fonctions objectif (optimisation multi-critère) et enfin d'autres peuvent être simulées par des stratégies de recherche lorsqu'elles expriment plus un but qu'une contrainte formellement définie.
3. Proposer des mesures de violation non plus guidées par un besoin sémantique mais plutôt dans le but de favoriser la convergence rapide des algorithmes de recherche locale vers de bonnes solutions. Ce thème nécessite de prendre en compte deux aspects fondamentaux : (1) savoir combiner des mesures portant sur des contraintes hétérogènes, et (2) définir des mesures adaptées à un dosage adéquat de la diversification par rapport à l'intensification.

# VI Caractérisation des solutions en optimisation

*Nous avons évoqué dans les chapitres précédents des problèmes d'optimisation qui comportent des contraintes métier sur les variables impliquées dans le critère à optimiser. La PPC est une théorie particulièrement adaptée à ce type de problèmes, car elle est relativement robuste à l'ajout de contraintes annexes et son pouvoir expressif est important. En outre, l'utilisation de techniques de recherche incomplètes lui permet souvent un passage à l'échelle tout à fait suffisant.*
*Pourtant, au début des années deux mille, relativement peu d'étude génériques de ce type de contraintes avaient été réalisées. Je me suis donc intéressé à ces contraintes, portant sur des variables impliquées dans les critères d'optimisation, ainsi qu'à leurs algorithmes de filtrage.*
*Ces algorithmes de filtrage doivent être légers pour avoir un intérêt pratique : complexité idéalement linéaire, simplicité des structures de données, convergence en une passe. Toute la difficulté du sujet se situe donc, à nouveau, dans la juste mesure à trouver entre le pouvoir expressif de ces contraintes, la puissance du filtrage des propagateurs associés à ces contraintes et le coût opérationnel des algorithmes.*
*Ces travaux m'ont amené à définir, de façon connexe, des contraintes dont la vocation est d'améliorer la résolution dans le cadre de problèmes d'optimisation. Elles sont brièvement présentées dans ce chapitre.*

*Collaborations : Pr. Nicolas Beldiceanu, Pr. Mats Carlsson, Dr. Xavier Lorca, Dr. Nina Narodytska, Pr. Jean-Charles Régin, PhD. Mohamed Siala, Pr. Toby Walsh.*

*Publications majeures : CPAIOR2010, IEEE-ICTAI2010, CP2011, IJAIT2011, IJCAI2011, CP2012, ECAI2012, IJCAI2013.*

## 1 Introduction

Dans le contexte des problèmes sur-contraints, nous avions évoqué l'intérêt de répartir de façon homogène les coûts de violation entre différentes composantes d'un même problème [121]. Ce besoin était apparu dans le cadre d'un projet européen (ECPLAIN) auquel j'ai participé pendant ma thèse, plus précisément concernant un problème de planification de coupe de bois en Norvège, où différents acteurs intervenaient avec des requêtes contradictoires : contraintes économiques, écologiques, découpage administratif des zones, routes, facilité accès aux forêts, etc.

Cependant, un tel besoin n'a rien de spécifique au cas des problèmes sur-contraints.

Nous verrons dans les exemples motivant les définitions des contraintes présentées dans la prochaine section que de telles contraintes sont présentes dans des problèmes d'optimisation dans différents contextes. L'intérêt d'utiliser une contrainte pour représenter des concepts génériques tels que la répartition homogène de valeurs, ou à l'inverse la concentration de valeurs élevées dans des sous-séquences précises de variables, ou encore le lissage d'un profil cumulatif, est double :

1. Une plus grande précision sémantique : il est possible d'exprimer des concepts difficiles à représenter par l'intermédiaire d'une fonction objectif, notamment lorsque ceux-ci sont de nature topologique.
2. Une meilleure efficacité de résolution : propager de telles contraintes à l'aide d'algorithmes dédiés peut accélérer considérablement le processus de résolution.

Ce chapitre présente des familles de contraintes dédiées à la caractérisation sémantique des solutions de problèmes d'optimisation, que j'ai réalisées seul ou en collaboration avec d'autres chercheurs.

De façon générale, nous pouvons poser le problème ainsi : une fonction objectif, qui représente un critère d'optimisation, peut être vue comme une agrégation de sous-objectifs qui sont représentés par des variables particulières, que nous appellerons variables de coût. En outre, des contraintes complémentaires à l'objectif permettent de caractériser les propriétés que doivent satisfaire les solutions pour pouvoir être mises en pratique. Ce sont ces contraintes qui nous intéressent dans ce chapitre. Il est intéressant d'enrichir les moteurs de résolution avec des contraintes *génériques*, qui expriment ces concepts communs à diverses applications.

## 2 Des contraintes dédiées à l'optimisation

### 2.1 La contrainte OrderedDistribute

**Concept.** Dans certains problèmes des valeurs de seuil sont définies, afin de limiter le nombre de variables prenant une valeur au-dessus de chaque seuil. D'un côté, les contraintes de *balancing*, e.g., Deviation [138], ne sont pas adaptées car elles traitent globalement l'ensemble des valeurs. D'un autre côté, il est dans la plupart des cas impossible d'utiliser des contraintes de cardinalité classiques car dans ces dernières il n'y a pas de lien entre le nombre d'occurrences d'une valeur $v$ et le nombre d'occurrences des valeurs strictement plus grandes que $v$. Enfin, une stratégie de choix de valeurs n'apportera aucune garantie dans ce contexte. Nous avons proposé une nouvelle contrainte, OrderedDistribute [118, 119], pour couvrir ce besoin. Notons qu'une autre alternative aux contraintes arithmétiques, différente et pertinente, a été publiée par la suite [29].

**Définition 12** *Soeint $X$ un ensemble de variables et $T$ un tableau de valeurs croissantes qui peuvent être assignées à des variables de $X$, tel que $|T| \geq 2$. Nous définissons $I_{max}$, un tableau d'entiers tel que $I_{max}[i]$ corresponde exclusivement à la valeur $T[i]$, et tel que $\forall i \in \{1, \ldots, |T|-1\}$, $I_{max}[i-1] \geq I_{max}[i]$.*
*Une affectation de valeurs $A(X)$ aux variables $X$ satisfait la contrainte* OrderedDistribute($X$, $T$, $I_{max}$) *si et seulement si les deux conditions suivantes sont satisfaites par $A(X)$ :*

1. *$\forall i \in \{0, 1, \ldots, |T|-1\}$, le nombre de valeurs $v$ : $v \geq T[i]$ est au plus $I_{max}[i]$.*

*2. Le nombre d'apparitions de la valeur $T[0]$ est au moins $|X| - I_{max}[1]$.*

**Applications.**

*1. Ordonnancement.* La contrainte OrderedDistribute est utile pour représenter des contraintes sur le profil de consommation d'activités consommant ou produisant de la ressource. Considérons par exemple un problème CuSP sur-contraint. Celui-ci peut être représenté en utilisant la contrainte SoftCumulative [116, 117, 40] qui modélise des dépassements de capacité sur des intervalles de temps fixes par des variables de coût. Supposons que la ressource soit humaine (le nombre d'employés d'une équipe de travail) et que chaque intervalle représente une heure. Chaque jour, chaque personne effectue un certain nombre d'activités. Le critère d'optimisation consiste à minimiser la somme des dépassements, mais cela n'est pas suffisant pour évaluer la qualité des solutions. Dans le contexte où les employés présents doivent absorber une possible surcharge de travail, il convient de la répartir sur la semaine, pour limiter le risque de voir certaines activités non terminées à temps ou mal réalisées. Plus un dépassement de la capacité en ressource est grand, plus il doit être rare dans une journée.

Pour modéliser un tel problème, on peut utiliser une décomposition à l'aide de la contrainte GlobalCardinality [132], et de contraintes de somme. Cette approche n'est pas bonne en termes de résolution, notamment parce qu'il est possible de prouver qu'effectuer un filtrage de toutes les valeurs non viables est NP-Difficile [118].

Avec notre approche, il suffit d'ajouter au modèle CuSP la contrainte OrderedDistribute.

*2. Autres applications.* Cette contrainte modélise des problèmes de chargement où des objets doivent être placés dans des containers et des containers dans des cargos. Une répartition peut être nécessaire basée sur le degré de fragilité des objets, afin de limiter les conséquences financières de containers endommagés. J.-C. Régin a également identifié un besoin similaire à celui de l'exemple du CuSP vu précédemment dans le cas d'un problème d'affectation de personnel. Plus généralement, la contrainte OrderedDistribute est une alternative aux contraintes de balancing utile lorsque des règles précises régissent les valeurs prises par les variables, selon un ordre total.

**Propagateur.** Nous avons proposé un propagateur GAC en $O(|X| + |T|)$. Cet algorithme converge en une passe et ne nécessite aucune structure de donnée autre que des tableaux indexés. Il a donc les propriétés requises pour un propagateur destiné à porter sur les variables impliquées dans un critère d'optimisation. À titre indicatif, nous avons extrait de l'article [119] la table comparant les performances du propagateur à celles d'une approche par décomposition.

| | Décomposition | OrderedDistribute |
|---|---|---|
| Obj | #Backtracks / Temps / Peuve | #Backtracks / Temps / Preuve |
| 47 | - / > 10 min / - | **963 / 4 s / oui** |
| 13 | **122 / 0 s / oui** | 50027 / 59 s / oui |
| 31 | 18550 / 2 min 52 s / oui | **370 / 5s / oui** |
| 16 | **44 / 0 s / oui** | 2562 / 11 s / oui |
| 9 | - / > 10 min / - | **448192 / 10 min / non** |
| 56 | - / > 10 min / - | **529 / 1 s / oui** |
| 19 | **26 / 0 s / oui** | 1361 / 8 s / oui |
| 2 | 1819 / 5 s / oui | 965 / 7 s / oui |
| 5 | 12251 / 14 s / oui | **665 / 10 s / oui** |
| 42 | - / > 10 min / - | **5406 / 9s / oui** |
| 13 | 33484 / 38 s / oui | **4274 / 12 s / oui** |
| unsat. | - / > 10 min / - | - / > 10 min / - |
| 17 | - / > 10 min / - | **533526 / 10 min / non** |
| 48 | - / > 10 min / - | **772 / 3 s / oui** |
| 33 | - / > 10 min / - | **860 / 3 s / oui** |
| 56 | - / > 10 min / - | **246 / 1 s / oui** |
| 46 | - / > 10 min / - | **546 / 3 s / oui** |
| 14 | 54946 / 62 s / oui | **1116 / 8 s / oui** |
| 43 | - / > 10 min / - | **615 / 3 s / oui** |
| unsat. | - / > 10 min / - | - / > 10 min / - |

Table 6 – Comparaison de la contrainte OrderedDistribute et de sa décomposition (backtracks, temps, preuve d'optimalité) sur des instances à de CuSP avec dépassements de ressource impliquant $n = 55$ activités [119].

### 2.2 La contrainte Focus et ses généralisations

**Concept.** Cette contrainte modélise le concept inverse des contraintes de balancing ou de la contrainte OrderedDistribute présentée précédemment. Dans certaines applications, les valeurs de coût élevées doivent être concentrées dans des zones topologiques précises et en nombre limité. Bien que la PPC semble un candidat particulièrement adapté (car, par exemple, ce concept est difficilement représentable par une fonction d'optimisation), il n'existait pas d'approches en PPC permettant de capturer correctement ce concept. J'ai donc introduit la contrainte Focus [112], qui couvre ce besoin. Nous utilisons les conventions suivantes. $X = [x_0, x_1, \ldots, x_{n-1}]$ est une séquence de variables, $s_{i,j}$ est une séquence d'indices de variables consécutives dans $X$, telle que $s_{i,j} = [i, i+1, \ldots, j]$, $0 \leq i \leq j < n$. $|E|$ désigne la taille d'une collection $E$.

**Définition 13 (Focus)** *Soit $y_c$ une variable, et $k$ et $len$ deux entiers, $1 \leq len \leq |X|$. Une affectation de valeurs aux variables $X \cup \{y_c\}$ satisfait* Focus*$(X, y_c, len, k)$ si et seulement si il existe un ensemble $S_X$ de séquences* disjointes *d'indices $s_{i,j}$ telle que les trois contraintes suivantes sont toutes satisfaites.*

1. $|S_X| \leq y_c$

2. $\forall x_l \in X,\ x_l > k \Leftrightarrow \exists s_{i,j} \in S_X$ *telle que* $l \in s_{i,j}$

3. $\forall s_{i,j} \in S_X,\ j - i + 1 \leq len$

Guidés par le souci de couvrir un plus large spectre d'applications, nous avons ensuite proposé trois généralisations de Focus [103].

La définition 13 impose que chaque séquence dans $S_X$ contienne exclusivement des valeurs $v > k$. Dans certains cas d'application, cette propriété est trop forte. Motivés par un exemple de *rental scheduling*, nous avons proposé de l'assouplir en ajoutant un paramètre $h$, qui indique qu'au plus $h$ valeurs plus petites que $k$ peuvent apparaître dans chaque séquence de $S_X$.

**Définition 14 (**SpringyFocus**)** *Soit $y_c$ une variable, et $k$, $len$ et $h$ trois entiers, $1 \leq len \leq |X|$, $0 \leq h < len - 1$. Une affectation de valeurs aux variables $X \cup \{y_c\}$ satisfait* SpringyFocus*$(X, y_c, len, h, k)$ si et seulement si il existe un ensemble $S_X$ de séquences* disjointes *d'indices $s_{i,j}$ telle que les quatre contraintes suivantes sont toutes satisfaites.*

1. $|S_X| \leq y_c$

2. $\forall x_l \in X,\ x_l > k \Rightarrow \exists s_{i,j} \in S_X$ *telle que* $l \in s_{i,j}$

3. $\forall s_{i,j} \in S_X,\ j - i + 1 \leq len,\ x_i > k$ *and* $x_j > k$.

4. $\forall s_{i,j} \in S_X,\ |\{l \in s_{i,j},\ x_l \leq k\}| \leq h$

Nous avons aussi proposé la contrainte WeightedFocus, qui étend Focus avec une variable $z_c$ qui permet de limiter la somme des longueurs des séquences dans $S_X$.

**Définition 15 (**WeightedFocus**)** *Let Soit $y_c$ une variable, $k$ et $len$ deux entiers, $1 \leq len \leq |X|$, et $z_c$ une variable. Une affectation de valeurs aux variables $X \cup \{y_c\} \cup \{z_c\}$ satisfait* WeightedFocus*$(X, y_c, len, k, z_c)$ si et seulement si il existe un ensemble $S_X$ de séquences* disjointes *d'indices $s_{i,j}$ telle que les quatre contraintes suivantes sont toutes satisfaites.*

1. $|S_X| \leq y_c$

2. $\forall x_l \in X,\ x_l > k \Leftrightarrow \exists s_{i,j} \in S_X$ *telle que* $l \in s_{i,j}$

3. $\forall s_{i,j} \in S_X,\ j - i + 1 \leq len$

4. $\sum_{s_{i,j} \in S_X} |s_{i,j}| \leq z_c$.

Enfin, nous avons proposé WeightedSpringyFocus, qui combine les deux extensions de Focusdes définitions 14 et 15.

**Applications.** Dans les problèmes de *rental scheduling*, les machines doivent être louées par forfait, sur des périodes de temps continues mais de durée limitée.

Par exemple, louer une machine pendant cinq semaines consécutives peut s'avérer plus économique que de réaliser cinq contrats de locations d'une semaine, séparés dans le temps. Cependant, si on en a besoin pendant six semaines consécutives, le loueur nous impose pour des raisons techniques de signer deux contrats, car ses offres ne dépassent pas cinq semaines.

Un autre domaine d'application est l'aide à la composition en musique, où l'on peu vouloir concentrer les violations de règles dans des zones précises de la partition générée, plutôt que de les répartir.

Enfin, des problèmes d'affectation peuvent être modélisés par Focus et ses généralisations, par exemple des contraintes annexes dans des problèmes de sport league scheduling [103].

| Instances | | | Focus($X, y_c, len, k$) | | | | Checker($X, y_c, len, k$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| nb. of chords | $\overline{y_c}$-$len$ -$k$ -$nmax$ | #optimum with $sum > 0$ | average #backtracks (of 100) | average #fails (of 100) | max. #backtracks (of 100) | average time (ms) (of 100) | average #backtracks (of 100) | average #fails (of 100) | max. #backtracks (of 100) | average time (ms) (of 100) |
| 8 | 1-4-0-3 | 66 | 61 | 46 | 462 | 11 | 1518 | 951 | 15300 | 17 |
| 8 | 1-4-1-3 | 66 | 45 | 33 | 342 | 1 | 91 | 59 | 1724 | 2 |
| 8 | 2-4-0-3 | 66 | 47 | 34 | 247 | 1 | 58 | 42 | 455 | 1 |
| 8 | 2-4-1-3 | 66 | 45 | 33 | 301 | 1 | 44 | 32 | 349 | 1 |
| 8 | 1-6-0-4 | 84 | 198 | 141 | 2205 | 12 | 15952 | 10040 | 86778 | 213 |
| 8 | 1-6-1-4 | 84 | 114 | 79 | 767 | 3 | 1819 | 1117 | 45171 | 24 |
| 8 | 2-6-0-4 | 84 | 127 | 88 | 620 | 3 | 2069 | 1361 | 64509 | 28 |
| 8 | 2-6-1-4 | 84 | 118 | 81 | 724 | 3 | 250 | 168 | 7027 | 4 |
| 8 | 1-8-0-5 | 98 | 261 | 184 | 1533 | 6 | 39307 | 25787 | 167575 | 566 |
| 8 | 1-8-1-5 | 98 | 148 | 103 | 662 | 3 | 11821 | 7642 | 94738 | 168 |
| 8 | 2-8-0-5 | 98 | 164 | 113 | 803 | 4 | 21739 | 14400 | 173063 | 317 |
| 8 | 1-8-0-5 | 98 | 183 | 127 | 882 | 4 | 10779 | 6939 | 92560 | 153 |
| 8 | 1-8-0-6 | 99 | 290 | 203 | 1187 | 18 | 46564 | 30488 | 130058 | 690 |
| 8 | 1-8-1-6 | 99 | 238 | 166 | 1167 | 11 | 29256 | 19150 | 134882 | 438 |
| 8 | 2-8-0-6 | 99 | 221 | 152 | 1458 | 6 | 29455 | 19607 | 123857 | 445 |
| 8 | 2-8-1-6 | 99 | 209 | 144 | 1118 | 9 | 21332 | 14095 | 117768 | 329 |
| 9 | 1-9-0-4 | 88 | 415 | 299 | 4003 | 18 | 214341 | 133051 | 1095734 | 3244 |
| 9 | 1-9-1-4 | 88 | 268 | 185 | 2184 | 6 | 12731 | 7988 | 751414 | 203 |
| 9 | 2-9-0-4 | 88 | 270 | 188 | 2714 | 6 | 22107 | 14065 | 374121 | 337 |
| 9 | 2-9-1-4 | 88 | 266 | 182 | 3499 | 6 | 1364 | 941 | 92773 | 23 |
| 9 | 1-9-0-5 | 97 | 574 | 407 | 2437 | 26 | 360324 | 230167 | 1355934 | 6584 |
| 9 | 1-9-1-5 | 97 | 404 | 273 | 1677 | 11 | 62956 | 40277 | 881441 | 1150 |
| 9 | 2-9-0-5 | 97 | 451 | 309 | 3327 | 12 | 228072 | 147007 | 1124630 | 4263 |
| 9 | 2-9-1-5 | 97 | 386 | 260 | 1698 | 10 | 58421 | 37589 | 989900 | 1079 |

Figure 12 – Tableau extrait de l'article CP2012 [112]. Il illustre l'intérêt de la propagation de la contrainte Focus associée à un critère d'optimisation consistant à minimiser les erreurs dans un problème musical (sorting chords problem) en comparant l'algorithme GAC (Focus) à une approche naïve (Checker). Une instrumentalisation simple de la fonction objectif, sans propagateur, a des conséquences négatives importantes sur le processus de résolution. Les résultats sont des moyennes sur 100 instances.

**Propagateurs.** Nous avons proposé des propagateurs effectuant la GAC, convergeant en une passe et ne nécessitant aucune structure de donnée incrémentale, avec les complexités temporelles suivantes :

- Focus : $O(|X|)$
- SpringyFocus : $O(|X|)$
- WeightedFocus : $O(|X| \times \overline{z_c})$
- WeightedSpringyFocus : $O(|X| \times \overline{z_c})$

| | 16_1 | | | 16_2 | | | 16_3 | | |
|---|---|---|---|---|---|---|---|---|---|
| | #n | #b | T | #n | #b | T | #n | #b | T |
| F | 50 | 0.9K | 0 | **50** | 4.1K | 2 | **47** | 18.1K | 7 |
| $D_1$ | 50 | 3.4K | 1 | 49 | 8.1K | 3 | 44 | 21.8K | 8 |

| | 20_1 | | | 20_2 | | | 20_3 | | |
|---|---|---|---|---|---|---|---|---|---|
| | #n | #b | T | #n | #b | T | #n | #b | T |
| F | **49** | 11.8K | 7 | **45** | 24.9K | 14 | **39** | 36.5K | 23 |
| $D_1$ | 43 | 30.8K | 13 | 35 | 27.2K | 12 | 29 | 29.6K | 17 |

FIGURE 13 – Tableau extrait de l'article IJCAI2013 [103]. Il illustre l'intérêt de la propagation de la contrainte WEIGHTEDFOCUS dans le cadre d'un problème de "sport league scheduling" [133] avec des contraintes annexes portant sur les droits télévisuels. $\#n$ désigne le nombre d'instances non résolues sur les 50 lancées (avec une limite de backtracks imposée à 400 000 au plus). $\#b$ est la moyenne de backtracks parmi les instances résolues et $T$ est la moyenne de temps parmi les instances résolues. La technique résolvant le plus d'instances peut donc être pénalisée par celles qui sont les plus difficiles. Les championnats comportent jusqu'à 20 équipes et la demi-saison 19 journées. La comparaison est faite entre WEIGHTEDFOCUS (F) une décomposition de WEIGHTEDFOCUS ($D_1$) proposée dans l'article IJCAI2013.

Nous avons réalisé plusieurs expérimentations, en ordonnancement et sur des problèmes musicaux. Les tables des figures 12 et 13 sont extraites des publications CP2012 et IJCAI2013.

Elles illustrent l'intérêt de définir des contraintes munies de leur propre propagateur, en complément de fonctions objectif qui peuvent alors rester assez simples, par exemple la minimisation d'une somme.

### 2.3 Des contraintes pour casser les symétries

**Concept.** La suppression des symétries de variables peut s'avérer cruciale pour résoudre certains problèmes. Ce thème a été largement abordé dans la littérature [54, 124, 125, 126, 143, 104].

Dans ce cadre, nous avons étudié le cas de contraintes globales associées avec une chaine d'inégalités de la forme $x_0 \leq x_1 \leq \dots x_{n-1}$, où les $x_i$ sont des variables appartenant à une séquence ordonnée de façon statique, lors de la définition du réseau de contraintes. Nous avons proposé des variations des contraintes NVALUE et de la contrainte $\sum_{x_i \in X} x_i = s$, qui prennent en compte la chaîne d'inégalités.

**La contrainte INCREASINGSUM.**

**Définition 16 (INCREASINGSUM)** *Soit $X = [x_0, x_1, \dots, x_{n-1}]$ une séquence de variables, et $s$ une variable. La contrainte* INCREASINGSUM$(s, X)$ *est satisfaite si et seulement si* $\forall i \in \{0, 1, \dots n-2\}, x_i \leq x_{i+1} \wedge \sum_{x_i \in X} x_i = s$.

Nous avons proposé un algorithme effectuant une consistance aux bornes de type BC avec une complexité temporelle en $O(\Theta(|X|)$ pour la contrainte INCREASINGSUM [120].

**La contrainte** IncreasingNValue**.**

**Définition 17 (**IncreasingNValue**)** *Soit $X = [x_0, x_1, \ldots, x_{n-1}]$ une séquence de variables, et $n_X$ une variable. Une affectation de valeurs aux variables satisfait* IncreasingNValue$(n_X, X)$ *si et seulement si :*

1. *$n_X$ est égale aux nombres de valeurs distinctes affectées à des variables dans $X$,*
2. *$\forall i \in \{0, 1, \ldots n-2\}, x_i \leq x_{i+1}$.*

J'ai eu l'idée d'un algorithme de balayage pour réaliser un propagateur pour IncreasingNValue. En collaboration avec X. Lorca, nous avons montré que cet algorithme réalise un filtrage GAC. Avec N. Beldiceanu, nous avons proposé une implémentation dont la complexité temporelle est linéaire en la somme des tailles des domaines [19].

D'un point de vue théorique, ce résultat est intéressant car réaliser un filtrage GAC pour la contrainte NValue [106], qui n'impose que la première des deux conditions de la définition 17, est NP-Difficile.

En collaboration avec F. Hermenier, IncreasingNValue a été expérimentée avec succès [19], dans le cadre d'*Entropy* [67], l'outil qu'il a réalisé pour traiter des problèmes d'allocation de machine virtuelles à des nœuds d'un réseau.

### 2.4 La contrainte SeqBin : Un algorithme GAC générique

**Concept.** Nous avons généralisé l'idée de l'algorithme de filtrage de IncreasingNValue pour l'étendre à une plus grande famille de contraintes, qu'il est possible de décrire comme des contraintes où une variable particulière exprime le nombre de fois où une certaine propriété est vérifiée sur une séquence de variables. Pour ce faire, nous avons présenté une contrainte générique, ou "méta-contrainte", SeqBin [114, 115]. Elle est basée sur une généralisation de la notion de *stretch* introduite par G. Pesant [108], appelée *$C-$stretch* car elle est relative à n'importe quelle contrainte binaire $C$, alors que la notion de stretch considère uniquement la contrainte d'égalité.

**Définition 18 (*$C$*-stretch)** *Soit $A[X]$ une affectation de valeurs aux variables d'une séquence $X = [x_0, x_1, \ldots, x_{n-1}]$ et $C$ une contrainte binaire. On note $A[x_i]$ la valeur de la variable $x_i$ dans $A[X]$. La contrainte C-sequence $\mathcal{C}(I[X], C)$ est satisfaite si et seulement si :*

- *Soit $n = 1$,*
- *Soit $n > 1$ at $\forall k \in [0, n-2]$ $C(A[x_k], A[x_{k+1}])$ est satisfaite.*

*Un* $C$-stretch *de $A[X]$ est une séquence de variables $X' \subseteq X$ pour laquelle les deux conditions suivantes sont satisfaites :*

1. *C-sequence $\mathcal{C}(A[X'], C)$ est satisfaite,*
2. *$\forall X"$ tel que $X' \subset X" \subseteq X$ C-sequence $\mathcal{C}(A[X"], C)$ n'est pas satisfaite.*

L'intuition de la définition 18 est de considérer des sous-séquences de variables consécutives de longueur maximale, telles que la contrainte binaire $C$ soit satisfaite entre toute paire de variables consécutives. Cette généralisation de la notion de stretch nous a permis de définir SeqBin.

**Définition 19** *La contrainte* SeqBin$(N, X, C, B)$ *est définie sur une variable* $n_X$*, une séquence de* $n$ *variables* $X = [x_0, x_1, \ldots, x_{n-1}]$ *et deux contraintes binaires* $C$ *et* $B$*. Étant donnée une affectation de valeurs aux variables* $A[N, x_0, x_1, \ldots, x_{n-1}]$*,* SeqBin$(N, X, C, B)$ *est satisfaite si et seulement si pour tout* $i \in [0, n-2]$*, la contrainte* $A[x_i]\, B\, A[x_{i+1}]$ *est satisfaite et and* $n_X$ *est égale au nombre de* $C$-stretches *dans* $A[X]$*.*

**Applications.** La contrainte SeqBin permet d'exprimer plusieurs contraintes, en utilisant son propagateur générique. Parmi celles-ci, la contrainte IncreasingNValue présentée précédemment, peut être utilisée pour éliminer efficacement des symétries de variables.

Dans un tout autre contexte, SeqBin permet d'exprimer la contrainte Change, introduite dans le cadre de problèmes d'affectation de personnel [42].

**Définition 20 (**Change**)** *La contrainte* Change *est définie sur une variable* $n_X$ *et une séquence* $X = [x_0, x_1, \ldots, x_{n-1}]$*, avec en paramètre une contrainte binaire* $C \in \{=, \neq, <, >, \leq, \geq\}$*. Elle est satisfaite si et seulement si* $n_X$ *est égale au nombre de fois où la contrainte* $C$ *est satisfaite sur des variables consécutives dans* $X$*.*

La contrainte Smooth$(n_X, X)$ est une variante de Change$(n_X, X, C)$, où $x_i\, C\, x_{i+1}$ est définie par la contrainte $|x_i - x_{i+1}| > cst$, $cst \in \mathbb{N}$. Cette contrainte est utile pour limiter le nombre de variations trop importantes dans un profil cumulatif [15]. Smooth peut elle aussi être représentée par SeqBin.

**Propagateur.** Le propagateur de SeqBin permet d'obtenir un filtrage GAC selon le type de contraintes binaires utilisées en paramètre. Il nécessite d'être spécialisé du point de vue des structures de données employées, le coeur de l'algorithme étant commun à toutes les spécialisations. Relativement à l'état de l'art, la complexité en temps du propagateur de SeqBin par rapport aux résultats existants est meilleure. Par exemple, la GAC pour Change peut être obtenue en une complexité en temps de l'ordre de la somme des tailles des domaines, en comparaison avec une complexité en $O(nd^3)$ en utilisant Slide [28], $O(n^2d^2)$ en utilisant Regular [109], ou encore en $O(n^3m)$ avec l'algorithme ad hoc de Hellsten [65, page 57], où $n = |X|$, $d$ est la taille maximale d'un domaine d'une variable dans $X$ et $m$ est la taille de l'union des domaines des variables $X$. Nous pouvons noter que la somme des tailles des domaines est bornée supérieurement par $n \times d$. D'autres approches par décomposition, e.g., en utilisant CostRegular [47], ne permettent pas d'obtenir un filtrage GAC.

## 3 Portée des travaux

Bien qu'ils soient récents, les articles présentant les contraintes évoquées dans ce chapitre ont été repris, essentiellement pour généraliser ces contraintes. La contrainte SeqBin a été récemment revisitée [78] afin d'étendre le spectre des contraintes représentable pour lesquelles ont peut obtenir un propagateur GAC via SeqBin. Nous pouvons également noter des généralisations de notre approche sur les contraintes dédiées aux suppression de symétries [101, 43].

En outre, nous avons proposé une contrainte générique développée en suivant la même philosophie que SeqBin, dans le cas de la conjonction d'un AllDiff et d'une contrainte de type $f(x_0) \oplus f(x_1) \oplus \ldots \oplus f(x_n) \leq cst$, où $f$ est une fonction unaire monotone croissante et $cst$ est une constante [17]. Ces travaux peuvent être mis en relation avec des propositions récentes de techniques génériques pour traiter efficacement des conjonctions de contraintes, comme par exemple des approches PPC et SAT pour traiter le cas de plusieurs AllDiff qui s'intersectent [81].

# VII Bilan et conclusion

Ce mémoire a abordé mes contributions scientifiques sous l'angle du compromis qu'un chercheur en optimisation doit trouver entre efficacité de résolution, expressivité, généralité et reproductibilité des approches. Cet angle de vue m'a toujours guidé dans mes travaux. Il est donc tout à fait représentatif de ma vision de la recherche scientifique. J'ai présenté dans chaque chapitre la portée, les limites et les perspectives de mes principales contributions, que l'on peut synthétiser de la façon suivante.

- *Le traitement de l'incertitude dans les problèmes d'optimisation.*
  Nous avons attaqué ce thème sous un angle de vue original qui consiste à caractériser de façon déclarative et sémantique la notion de robustesse aux aléas, en fonction de chaque famille de problème.
- *L'aide à la modélisation.*
  J'ai présenté des contributions très variées : apprentissage pour améliorer des modèles à contraintes, génération de propagateurs à partir de représentations déclaratives des contraintes, automates, démocratisation des consistances fortes, formalisation précise du niveau de consistance des propagateurs dans le but de rendre leur déclenchement dynamique et transparent pour l'utilisateur.
- *Les problèmes sur-contraints.*
  À partir du sujet initial de ma thèse, consistant à modéliser un problème sur-contraint sous la forme d'un problème d'optimisation standard, j'ai proposé le nouveau thème des contraintes globales souples qui a eu une portée importante dans la communauté.
- *L'aide à la décision.*
  J'ai étendu certains résultats obtenus sur les problèmes sur-contraints au sujet plus général des problèmes d'optimisation en PPC, en contribuant à l'élaboration de techniques génériques pour exprimer des concepts complémentaires aux critères d'optimisation, qui permettent de réduire le fossé existant parfois entre les solutions fournies par un système et les solutions qui sont mises en œuvre en pratique.
- *Les algorithmes.*
  Au travers des différents thèmes présentés ci-dessus, j'ai conçu et implémenté plusieurs algorithmes et techniques de résolution, expérimentés dans des contextes d'applications variés. Parmi ces familles d'application, l'ordonnancement s'est avéré au fil des années mon thème prévilégié.

La PPC offre l'avantage de fournir un cadre formel extrêmement propre, reproductible, tout en permettant l'usage de techniques de résolution hétérogènes.

Son potentiel me semble encore trop méconnu dans l'industrie, d'une part parce que cette technologie demeure difficile d'accès pour un utilisateur non expert, et d'autre part, à mon avis, parce qu'elle souffre d'un défaut de communication concernant son potentiel pour résoudre des applications. Ce deuxième aspect est davantage de nature philosophique que scientifique et il comporte un danger : refuser toute idée trop novatrice pour pouvoir fonctionner immédiatement.

Une condition nécessaire à l'expansion du domaine me semble être d'accepter que la PPC a naturellement vocation a être hybridée et qu'il n'existe aucune vérité absolue concernant les bonnes et les mauvaises techniques. Par exemple, le fait de vouloir et de pouvoir traiter des problèmes de grande taille, peu combinatoires, n'implique pas de cesser d'étudier des problèmes plus petits et plus difficiles, car les besoins industriels évoluent parfois de façon peu prévisible. De même, l'importance de disposer d'outils de résolution très bien conçus ne s'oppose aucunement au besoin de contributions scientifiques génériques, qui considèrent les contraintes comme des objets mathématiques.

Il existe de nombreux autres exemples, e.g., approche variable vs CSP fonctionnels pour traiter les problèmes sur-contraints, techniques de recherche vs explications, propagateurs légers vs contraintes globales qui assurent un filtrage GAC, etc.

La PPC est naturellement à l'intersection entre la Recherche Opérationnelle et l'Intelligence Artificielle. Ce n'est pas un défaut, mais une force. La diversité des points de vue, favorisant l'idée qu'une recherche en amont et la recherche appliquée sont complémentaires et non pas en opposition, conduiront à une nouvelle génération d'outils de résolution de problèmes industriels, plus simples d'accès et réellement déclaratifs.

# Bibliographie


[1] O. Akgun, A. M. Frisch, B. Hnich, C. Jefferson, and I. Miguel. Conjure revisited : Towards automated constraint modelling. *CoRR*, abs/1109.1774, 2011.

[2] J. Amilhastre. Representation par automate d'ensembles de solutions de problemes de satisfaction de contraintes. *Ph.D dissertation*, 1999.

[3] P. Baptiste, C. Le Pape, and W. Nuijten. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92 :305–333, 1999.

[4] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling : Applying Constraint Programming to Scheduling Problems*. International Series in Operations Research and Management Science. Kluwer, 2001.

[5] P. Baptiste, C. Le Pape, and L. Peridy. Global constraints for partial CSPs : A case-study of resource and due date constraints. *Proc. CP*, pages 87–102, 1998.

[6] N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. *Proc. CP*, pages 52–66, 2000.

[7] N. Beldiceanu and M. Carlsson. Sweep as a generic pruning technique applied to constraint relaxation. *Proc. of the CP workshop SOFT'01 : Modeling and Solving Problems with Soft Constraints*, pages 43–55, 2001.

[8] N. Beldiceanu and M. Carlsson. A new multi-resource *cumulatives* constraint with negative heights. *Proc. CP*, pages 63–79, 2002.

[9] N. Beldiceanu, M. Carlsson, R. Debruyne, and **T. Petit**. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4) :339–362, 2005.

[10] N. Beldiceanu, M. Carlsson, S. Demassey, and **T. Petit**. Graph properties based filtering. In *CP*, pages 59–74, 2006.

[11] N. Beldiceanu, M. Carlsson, S. Demassey, and **T. Petit**. Global constraint catalogue : Past, present and future. *Constraints*, 12(1) :21–62, 2007.

[12] N. Beldiceanu, M. Carlsson, P. Flener, and J. Pearson. On matrices, automata, and double counting in constraint programming. *Constraints*, 18(1) :108–140, 2013.

[13] N. Beldiceanu, M. Carlsson, J. Rampon, and C. Truchet. Graph invariants as necessary conditions for global constraints. In *CP*, pages 92–106, 2005.

[14] N. Beldiceanu, M. Carlsson, and J-X. Rampon. Global constraint catalog. *SICS Technical report T2005-08, URL : http ://www.sics.se/libindex.html#Technical*, 2005.

[15] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global Constraint Catalog, 2nd Ed. Technical Report T2010-07, SICS, 2010.

[16] N. Beldiceanu, M. Carlsson, and **T. Petit**. Deriving filtering algorithms from constraint checkers. In *CP*, pages 107–122, 2004.

[17] N. Beldiceanu, M. Carlsson, **T. Petit**, and J.-C. Régin. An o(nlog n) bound consistency algorithm for the conjunction of an alldifferent and an inequality between a sum of variables and a constant, and its generalization. In *ECAI*, pages 145–150, 2012.

[18] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12) :97–123, 1994.

[19] N. Beldiceanu, F. Hermenier, X. Lorca, and **T. Petit**. The increasing nvalue constraint. In *CPAIOR*, pages 25–39, 2010.

[20] N. Beldiceanu and T. Petit. Cost evaluation of soft global constraints. In *CPAIOR*, pages 80–95, 2004.

[21] N. Beldiceanu, **T. Petit**, and G. Rochart. Bounds of graph characteristics. In *CP*, pages 742–746, 2005.

[22] N. Beldiceanu, **T. Petit**, and G. Rochart. Bounds of graph parameters for global constraints. *RAIRO - Operations Research*, 40(4) :327–353, 2006.

[23] C. Berge. Graphs and hypergraphs. *Dunod, Paris*, 1970.

[24] C. Bessiere. Constraint propagation. Research report 06020 (Chapter 3 of the Handbook of Constraint Programming, F. Rossi, P. van Beek and T. Walsh eds. Elsevier 2006.), LIRMM, 2006.

[25] C. Bessiere, R. Coletta, and **T. Petit**. Acquiring parameters of implied global constraints. *Proc. CP*, pages 747–751, 2005.

[26] C. Bessière, R. Coletta, and **T. Petit**. Learning implied global constraints. *Proc. IJCAI*, pages 44–49, 2007.

[27] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, C.-G. Quimper, and T. Walsh. Reformulating global constraints : The slideand regularconstraints. In *SARA*, pages 80–92, 2007.

[28] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Slide : A useful special case of the cardpath constraint. In *ECAI*, pages 475–479, 2008.

[29] C. Bessiere, E. Hebrard, G. Katsirelos, Z. Kiziltan, É. Picard-Cantin, C.-G. Quimper, and T. Walsh. The balance constraint family. In *CP*, pages 174–189, 2014.

[30] C. Bessiere, K. Stergiou, and T. Walsh. Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, 172(6-7) :800–822, 2008.

[31] C. Bessière, **T. Petit**, and B. Zanuttini. Making bound consistency as effective as arc consistency. In *IJCAI*, pages 425–430, 2009.

[32] C. Bessière and P. Van Hentenryck. To be or not to be... a global constraint. *Proc. CP*, pages 789–794, 2003.

[33] J.-C. Billaut, A. Moukrim, and E. Sanlaville, editors. *Flexibility and Robustness in Scheduling*. Wiley, 2010.

[34] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs : Frameworks, properties, and comparison. *Constraints*, 4 :199–240, 1999.

[35] J. Du Boisberranger, D. Gardy, X. Lorca, and C. Truchet. When is it worthwhile to propagate a constraint ? a probabilistic analysis of alldifferent. In *ANALCO*, pages 80–90, 2013.

[36] L. Bordeaux and E. Monfroy. Beyond NP : Arc-consistency for quantifed constraints. *CP*, pages 371–386, 2002.

[37] S. Brockbank, G. Pesant, and L.-M. Rousseau. Counting spanning trees to guide search in constrained spanning tree problems. In *CP*, pages 175–183, 2013.

[38] H. Cambazard, E. Hebrard, B. O'Sullivan, and A. Papadopoulos. Local search and constraint programming for the post enrolment-based course timetabling problem. *Annals OR*, 194(1) :111–135, 2012.

[39] Choco. 3.1.0. *URL: http://choco.sourceforge.net/*, 2013.

[40] A. De Clercq, **T. Petit**, N. Beldiceanu, and N. Jussien. Filtering algorithms for discrete cumulative problems with overloads of resource. *Proc. CP*, pages 240–255, 2011.

[41] S. Colton and I. Miguel. Constraint generation via automated theory formation. *Proc. CP*, pages 575–579, 2001.

[42] Cosytec. *CHIP Reference Manual*, release 5.1 edition, 1997.

[43] T.-B.-H. Dao, K.-C. Duong, and C. Vrain. A declarative framework for constrained clustering. In *ECML/PKDD (3)*, pages 419–434, 2013.

[44] A.-J. Davenport, C. Jefflot, and J.-C. Beck. Slack-based techniques for robust schedules. In *European Conference on Planning*, pages 7–18, 2001.

[45] R. Debruyne and C. Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230, 2001.

[46] R. Debruyne and C. Bessiere. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230, 2001.

[47] S. Demassey, G. Pesant, and L.-M. Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4) :315–333, 2006.

[48] A. Derrien and **T. Petit**. The energetic reasoning checker revisited. *CoRR*, abs/1310.5564, 2013.

[49] A. Derrien and **T. Petit**. A new characterization of relevant intervals for energetic reasoning. In *Proccedings of the 20th International Conference CP 2014, to appear*, Lecture Notes in Computer Science. Springer, 2014.

[50] A. Derrien, **T. Petit**, and S. Zampelli. A declarative paradigm for robust cumulative scheduling. In *the Proccedings of the 20th International Conference CP 2014, to appear*, Lecture Notes in Computer Science. Springer, 2014.

[51] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. *FUZZ-IEEE'93*, 1993.

[52] J.-G. Fages, X. Lorca, and **T. Petit**. Self-decomposable global constraints. In *Proccedings of ECAI 2014 International Conference, to appear*, 2014.

[53] H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. *First European Congress on Fuzzy and Intelligent Technologies*, 1993.

[54] P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *CP*, pages 462–476, 2002.

[55] E. C. Freuder. Partial constraint satisfaction. *Proc. IJCAI*, pages 278–283, 1989.

[56] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58 :21–70, 1992.

[57] A. M. Frisch, C. Jefferson, B. Martinez Hernandez, and Ian Miguel. The rules of constraint modelling. *Proc. IJCAI*, pages 109–116, 2005.

[58] M. R. Garey and D. S. Johnson. Computers and intractability : A guide to the theory of NP-completeness. *W.H. Freeman and Company*, ISBN 0-7167-1045-5, 1979.

[59] Luca Di Gaspero and Andrea Schaerf. Easylocal++ : an object-oriented framework for the flexible design of local-search algorithms. *Softw., Pract. Exper.*, 33(8) :733–765, 2003.

[60] D. Hanák. Implementing global constraints as structured graphs of elementary constraints. *Scientific Journal Acta Cybernetica*, 2003.

[61] E. Hebrard. Robust solutions for constraint satisfaction and optimization. In *AAAI*, pages 952–953, 2004.

[62] E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In *CPAIOR*, pages 157–172, 2004.

[63] E. Hebrard, D. Marx, B. O'Sullivan, and I. Razgon. Soft constraints of difference and equality. *J. Artif. Intell. Res. (JAIR)*, 41 :97–130, 2011.

[64] E. Hebrard, B. O'Sullivan, and I. Razgon. A soft constraint of equality : Complexity and approximability. In *CP*, pages 358–371, 2008.

[65] L. Hellsten. Consistency propagation for *stretch* constraints. Master's thesis, Waterloo, 2004.

[66] P. Van Hentenryck, Y. Deville, and C.M. Teng. A generic arc consistency algorithm and its specializations. *Artificial Intelligence*, 57(2-3) :291–321, 1992.

[67] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. L. Lawall. Entropy : a consolidation manager for clusters. In *VEE*, pages 41–50, 2009.

[68] W.-J. Van Hoeve, G. Pesant, and L.-M. Rousseau. On global warming : Flow-based soft global constraints. *Journal of Heuristics*, 12 :4-5 :475–489, 2006.

[69] J-L. de Siqueira N. and J-F. Puget. Explanation-based generalization of failures. *Proc. ECAI*, pages 339–344, 1988.

[70] P. Janssen and M-C. Vilarem. Problèmes de satisfaction de contraintes : techniques de résolution et application à la synthèse de peptides. *Research Report C.R.I.M.*, 54, 1988.

[71] P. Jégou. Contribution à l'étude des problèmes de satisfaction de contraintes : algorithmes de propagation et de résolution. Propagation de contraintes dans les réseaux dynamiques. *PhD Thesis*, 1991.

[72] N. Jussien. Relaxation de contraintes pour les problèmes dynamiques. *Ph.D. Dissertation, Université de Rennes I*, 1997.

[73] N. Jussien. The versatility of using explanations within constraint programming. *Habilitation thesis of Université de Nantes*, 18 September 2003. also available as RR-03-04 research report at École des Mines de Nantes.

[74] N. Jussien and P. Boizumault. Best-first search for property maintenance in reactive constraint systems. In *ILPS*, pages 339–353, 1997.

[75] S. Kadioglu and M. Sellmann. Efficient context-free grammar constraints. In *AAAI*, pages 310–316, 2008.

[76] R. Kameugne, L. P. Fotso, J. Scott, and Y. Ngo-Kateu. A quadratic edge-finding filtering algorithm for cumulative resource constraints. In *CP*, pages 478–492, 2011.

[77] G. Katsirelos, N. Narodytska, and T. Walsh. The weighted cfgconstraint. In *CPAIOR*, pages 323–327, 2008.

[78] G. Katsirelos, N. Narodytska, and T. Walsh. The seqbin constraint revisited. In *CP*, pages 332–347, 2012.

[79] A. Lahrichi. The notions of Hump, Compulsory Part and their use in Cumulative Problems. *C.R. Acad. sc.*, t. 294 :204–211, 1982.

[80] Arnaud Lallouet and Andrei Legtchenko. Partially defined constraints in constraint-based design. *AI EDAM*, 20(4) :297–311, 2006.

[81] F. Lardeux, E. Monfroy, and F. Saubion. Interleaved alldifferent constraints : CSP vs. SAT approaches. *Proc. AIMSA*, pages 380–384, 2008.

[82] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107 :149–163, 1999.

[83] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159(1-2) :1–26, 2004.

[84] J.-L. Laurière. Un langage et un programme pour énoncer et résoudre des problèmes combinatoires. *Thèse de Doctorat d'État de l'université Paris 6*, 1976.

[85] J.-L. Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1) :29–127, 1978.

[86] N. Lazaar, A. Gotlieb, and Y. Lebbah. On testing constraint programs. In *CP*, pages 330–344, 2010.

[87] J. H.-M. Lee and K. L. Leung. Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In *IJCAI*, pages 559–565, 2009.

[88] M. Lemaitre, G. Verfaillie, E. Bourreau, and F. Laburthe. Integrating algorithms for weigthed csp in a constraint programming framework. *CP Worshop on Modelling and Solving Problems with Soft Constraints*, 2001.

[89] K. Leo, C. Mears, G. Tack, and M. Garcia de la Banda. Globalizing constraint models. In *CP*, pages 432–447, 2013.

[90] D. Lesaint, D. Mehta, B. O'Sullivan, L. Quesada, and N. Wilson. A soft global precedence constraint. In *IJCAI*, pages 566–571, 2009.

[91] A. Letort. Passage à l'échelle pour les contraintes d'ordonnancement multi-ressources. *Ph.D dissertation*, 2013.

[92] A. Letort, N. Beldiceanu, and M. Carlsson. A scalable sweep algorithm for the cumulative constraint. In *CP*, pages 439–454, 2012.

[93] R. Leus, C. Artigues, and F. Talla Nobibon. Robust optimization for resource-constrained project scheduling with uncertain activity durations. In *IEEM*, pages 101–105, 2011.

[94] P. Lopez, J. Erschler, and P. Esquirol. Ordonnancement de tâches sous contraintes : une approche énergétique. *Automatique, Productique, Informatique Industrielle*, 26 :5-6 :453–481, 1992.

[95] A. López-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *IJCAI*, pages 245–250, 2003.

[96] M. J. Maher. Contractibility and contractible approximations of soft global constraints. In *ICLP (Technical Communications)*, pages 114–123, 2010.

[97] L. Mercier and P. Van Hentenryck. Strong polynomiality of resource constraint propagation. *Discrete Optimization*, 4(3-4) :288–314, 2007.

[98] L. Mercier and P. Van Hentenryck. Edge finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1) :143–153, 2008.

[99] P. Meseguer, J. Larrosa, and M. Sanchez. Lower bounds for non-binary constraint optimization problems. *Proc. CP*, pages 317–331, 2001.

[100] J.-P. Métivier, P. Boizumault, and S. Loudni. All different : Softening alldifferent in weighted csps. In *ICTAI (1)*, pages 223–230, 2007.

[101] J.-N. Monette, N. Beldiceanu, P. Flener, and J. Pearson. A parametric propagator for discretely convex pairs of sum constraints. In *CP*, pages 529–544, 2013.

[102] U. Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Information Science*, 7 :95–132, 1974.

[103] N. Narodytska, **T. Petit**, M. Siala, and T. Walsh. Three generalizations of the focus constraint. In *IJCAI*, 2013.

[104] N. Narodytska and T. Walsh. Breaking symmetry with different orderings. In *CP*, pages 545–561, 2013.

[105] W. Nuijten. *Time and resource constrained scheduling : a Constraint Satisfaction Approach.* Doctoral dissertation, Eindhoven University of Technology, 1994.

[106] F. Pachet and P. Roy. Automatic generation of music programs. *Proc. CP*, pages 331–345, 1999.

[107] L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. In *CP*, pages 468–481, 2004.

[108] G. Pesant. A filtering algorithm for the stretch constraint. *Proc. CP*, pages 181–195, 2001.

[109] G. Pesant. A regular language membership constraint for finite sequences of variables. *Proc. CP*, 3258 :482–495, 2004.

[110] G. Pesant and J.-C. Régin. Spread : A balancing constraint based on statistics. *Proc. CP*, pages 460–474, 2005.

[111] T. Petit. *Modélisation et algorithmes de résolution des problèmes sur-contraints.* Doctoral dissertation, Université Montpellier II, 2002.

[112] T. Petit. Focus : A constraint for concentrating high costs. In *CP*, pages 577–592, 2012.

[113] T. Petit. Intermediary local consistencies. In *ECAI*, pages 919–920, 2012.

[114] T. Petit, N. Beldiceanu, and X. Lorca. A generalized arc-consistency algorithm for a class of counting constraints. In *IJCAI*, pages 643–648, 2011.

[115] T. Petit, N. Beldiceanu, and X. Lorca. A generalized arc-consistency algorithm for a class of counting constraints : Revised edition that incorporates one correction. *CoRR*, abs/1110.4719, 2011.

[116] T. Petit and E. Poder. Global propagation of practicability constraints. *Proc. CPAIOR*, 5015 :361–366, 2008.

[117] T. Petit and E. Poder. Global propagation of side constraints for solving over-constrained problems. *Annals of 0perations Research*, pages 295–314, 2011.

[118] T. Petit and J.-C. Régin. The ordered distribute constraint. In *ICTAI*, pages 431–438, 2010.

[119] T. Petit and J.-C. Régin. The ordered distribute constraint. *International Journal on Artificial Intelligence Tools*, 20(4) :617–637, 2011.

[120] T. Petit, J.-C. Régin, and N. Beldiceanu. A $\theta$(n) bound-consistency algorithm for the increasing sum constraint. In *CP*, pages 721–728, 2011.

[121] T. Petit, J-C. Régin, and C. Bessière. Meta constraints on violations for over constrained problems. *Proc. IEEE-ICTAI*, pages 358–365, 2000.

[122] T. Petit, J-C. Régin, and C. Bessière. Specific filtering algorithms for over constrained problems. *Proc. CP*, pages 451–463, 2001.

[123] T. Petit, J-C. Régin, and C. Bessière. Range-based algorithm for Max-CSP. *ECAI workshop on Modelling and Solving Problems with Constraints*, 2002.

[124] J.-F. Puget. Symmetry breaking revisited. In *CP*, pages 446–461, 2002.

[125] J.-F. Puget. Symmetry breaking using stabilizers. In *CP*, pages 585–599, 2003.

[126] Jean-François Puget. Automatic detection of variable and value symmetries. *Proc. CP*, pages 475–489, 2005.

[127] J.F. Puget. Constraint programming next challenge : Simplicity of use. *Proc. CP*, pages 5–8, 2004.

[128] C.-G. Quimper and T. W. Decomposing global grammar constraints. In *CP*, pages 590–604, 2007.

[129] C.-G. Quimper and T. W. Decompositions of grammar constraints. In *AAAI*, pages 1567–1570, 2008.

[130] C.-G. Quimper and T. Walsh. Global grammar constraints. In *CP*, pages 751–755, 2006.

[131] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. *Proc. AAAI*, pages 362–367, 1994.

[132] J-C. Régin. Generalized arc consistency for global cardinality constraint. *Proc. AAAI*, pages 209–215, 1996.

[133] J-C. Régin. Minimization of the number of breaks in sports scheduling problems using constraints programming. *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 57 :115–130, 2001.

[134] J-C. Régin, J-F. Puget, and **T. Petit**. Representation of soft constraints by hard constraints. *Proc. JFPLC'02*, 2002.

[135] J.-C. Régin and **T. Petit**. The objective sum constraint. In *CPAIOR*, pages 190–195, 2011.

[136] J-C. Régin, **T. Petit**, C. Bessière, and J-F. Puget. An original constraint based approach for solving over constrained prolems. *Proc. CP*, pages 543–548, 2000.

[137] J-C. Régin, **T. Petit**, C. Bessière, and J-F. Puget. New lower bounds of constraint violations for over constrained prolems. *Proc. CP*, pages 332–345, 2001.

[138] P. Schaus, Y. Deville, P. Dupont, and J-C. Régin. The deviation constraint. *Proc. CPAIOR*, 4510 :260–274, 2007.

[139] P. Schaus, P. Van Hentenryck, and J-C. Régin. Scalable load balancing in nurse to patient assignment problems. *Proc. CPAIOR*, 5547 :248–262, 2009.

[140] T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints ?". *Proc. of 8th Conf. of Uncertainty in AI*, 1992.

[141] C. Schulte and G. Tack. Perfect derived propagators. In *CP*, pages 571–575, 2008.

[142] B. Smith, K. Stergiou, and T. Walsh. Modelling the golomb ruler problem. *Proc. IJCAI'99 workshop on non-binary constraints*, 1999.

[143] B. M. Smith, S. Bistarelli, and B. O'Sullivan. Constraint symmetry for the soft csp. In *CP*, pages 872–879, 2007.

[144] G. Tack, C. Schulte, and G. Smolka. Generating propagators for finite set constraints. In *CP*, pages 575–589, 2006.

[145] Guido Tack. *Constraint Propagation – Models, Techniques, Implementation.* Doctoral dissertation, Saarland University, 2009.

[146] P. Torres and P. Lopez. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127(2) :332–343, 2000.

[147] P. Van Hentenryck and L. Michel. Control abstractions for local search. *Proc. CP*, pages 66–80, 2003.

[148] W.J. van Hoeve. A hyper-arc consistency algorithm for the soft alldifferent constraint. In *CP*, pages 679–689, 2004.

[149] N. R. Vempaty. Solving constraint satisfaction problems using finite state automata. In *National Conference on Artificial Intelligence (AAAI-92)*, pages 453–458. AAAI Press, 1992.

[150] P. Vilím. Max energy filtering algorithm for discrete cumulative resources. *Proc. CPAIOR*, 5547 :294–308, 2009.

[151] Petr Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In *CPAIOR*, pages 230–245, 2011.

[152] J. Vion, **T. Petit**, and N. Jussien. Integrating strong local consistencies into constraint solvers. In *CSCLP*, pages 90–104, 2009.

[153] C. Wei Wu, K. N. Brown, and J. C. Beck. Scheduling with uncertain durations : Modeling beta-robust scheduling with constraints. *Computers & OR*, 36(8) :2348–2356, 2009.

[154] S. Zampelli, Y. Vergados, R. Van Schaeren, W. Dullaert, and B. Raa. The berth allocation and quay crane assignment problem using a cp approach. In *CP*, pages 880–896, 2013.

[155] A. Zanarini, M. Milano, and G. Pesant. Improved algorithm for the soft global cardinality constraint. In *CPAIOR*, pages 288–299, 2006.

[156] A. Zanarini and G. Pesant. Solution counting algorithms for constraint-centered search heuristics. *Constraints*, 14(3) :392–413, 2009.

[157] Y. Zhang and R. H. C. Yap. Arc consistency on n-ary monotonic and linear constraints. *Proc. CP*, pages 470–483, 2000.

[158] M. Zytnicki, C. Gaspin, S. de Givry, and T. Schiex. Bounds arc consistency for weighted csps. *J. Artif. Intell. Res. (JAIR)*, 35 :593–621, 2009.